UNIVERSITETET I OSLO
Institutt for informatikk

Et program for konstruksjon og simulering av digitale kretser

Kandeeban Arumugam

Våren 2009

Et program for konstruksjon og simulering av digitale kretser

Kandeeban Arumugam

Våren 2009

Sammendrag

Denne oppgaven handler om utviklingen av et spesielt hjelpemiddel til undervisning i digitalteknikk: et program for konstruksjon og simulering av digitale kretser. Med dette kan studentene enkelt tegne små logiske kretser og simulere hvorledes de fungerer.

Programmet skal oppfylle visse egenskaper: Det må være meget enkelt å bruke. Det må være platformuavhengig siden universiteter og høgskoler disponerer datamaskiner med ulike operativsystemer; det samme gjelder de studentene som har private hjemmemaskiner. Programmets kildekode bør være tilgjengelig slik at den enkelte institusjon kan utvikle det videre i henhold til sine egne behov om nødvendig. Til sist vil det være en klar fordel i et undervisningsmiljø at programmer er gratis.

I dag finnes det, så vidt jeg vet, ingen programmer som tilfredsstiller alle kravene over. Derfor håper jeg programmet Digital Circuit vil være til nytte for mange.

Derom du er interessert i å prøve programmet, finnes informasjon om dette på http://ifi.uio.no/~kandeeba/.

Innhold

Figurer

Forord

Denne masteroppgaven er skrevet av Kandeeban Arumugam for graden «Master i informatikk» ved Institutt for informatikk ved Universitetet i Oslo. Veileder for oppgaven har vært universitetslektor Dag Langmyhr.

Leseren av denne oppgaven forvents å ha generell kompetanse innen IT, som for eksempel bachelorgrad. Leseren trenger ikke å vite mye om digitale kretser, fordi dette forklares i oppgaven.

Jeg vil takke flere personer som har bidratt for å få denne rapporten fullført. Først og fremst vil jeg takke universitetslektor Dag Langmyhr som har veiledet meg tålmodig gjennom hele prosjektet. Jeg vil også rette en stor takk til Hashmat Khan, Nurettin Erman og Sutharsan Tharmathas som har brukt tid til å lese gjennom oppgaven og kommet med mange gode forslag til endringer. Deres hjelp underveis har vært av helt uvurderlig verdi, jeg har alltid fått hjelp til å finne veien videre og ny inspirasjon til å fortsette. Tusen hjertelig takk for deres bidrag.

Denne oppgaven hadde ikke vært mulig uten min familie, og deres uvurderlige støtte under arbeidet med oppgaven. Jeg takker Selvarani Arumugam, Arumugam Kasipillai, Karthika Arumugam og Sundaralingam Kasipillai for at dere alltid var tilstedet når jeg hadde behov for støtte. Også må jeg takke min bror Sutharsan Arumugam, som gjorde den matematiske delen av oppgaven mye enklere for meg å forstå.

Jeg må også takke Pathmajeyam familien for alle de gode råd dere har gitt meg. Til slutt vil jeg også takke Tharany Pathmajeyam som også har bidratt med å rette oppgaven, og har alltid vært til stedet når jeg har hatt behov for omsorg og kjærlighet. Glad i deg!

Innledning

I dag gir mange høgskoler og universiteter kurs i digitalteknikk, hvor studentene må tegne digitale logiske kretser ved hjelp av et verktøy. Dette viser at behovet er tilstedet, og at det trengs et godt undervisningsverktøy som mange undervisningsinstitusjoner kan ha nytte av.

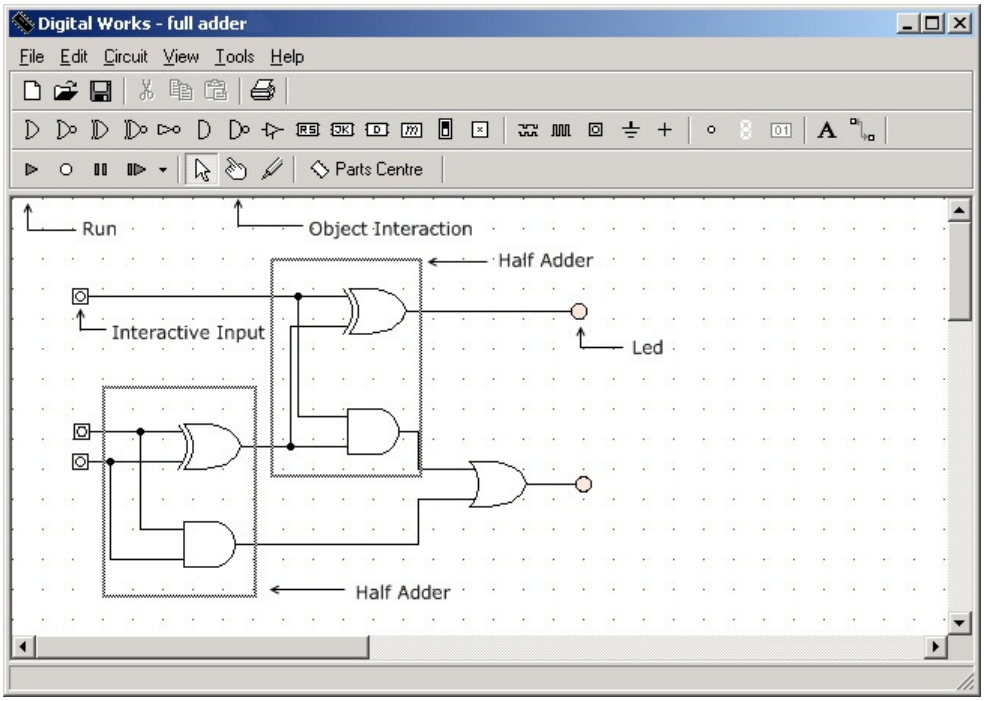
Digital Circuit er navnet på applikasjonen som jeg har utviklet med tanke på å dekke behovet til studenter som vil arbeide med digitale kretser i forbindelse med undervisningen. Navnet til applikasjonen er selvforklarende. Søk gjort på flere søkemotorer indikerer at det ikke eksiterer andre programmer med det samme navnet.

Verktøy

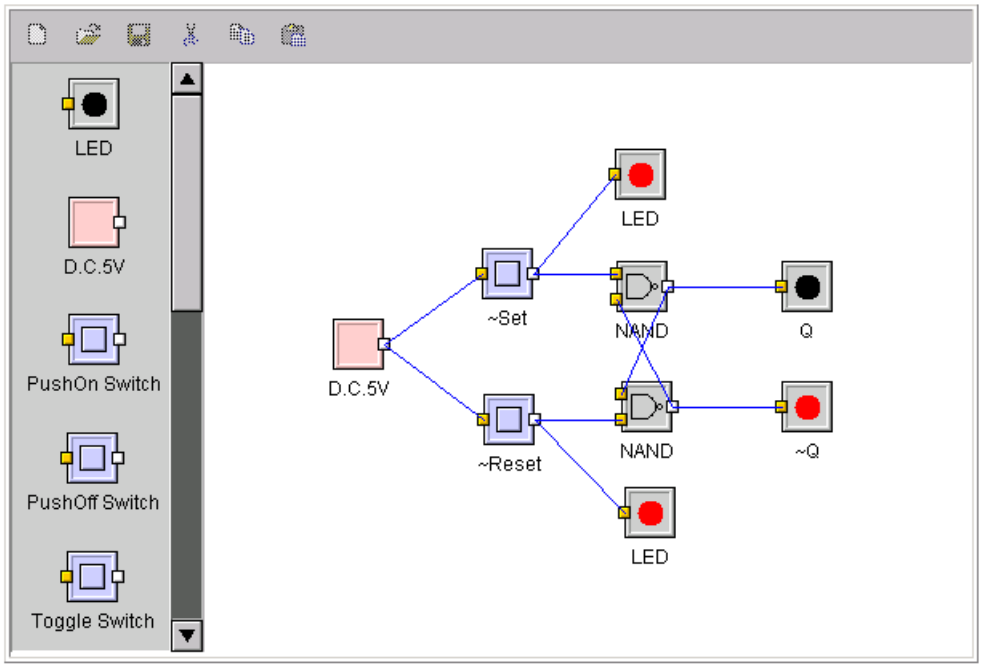
Det finnes mange utviklingsverktøy for å tegne logiske kretser, men ikke alle er like velegnet til undervisningsformål. Mange av disse er profesjonelle programmer som er ganske store, komplekse og dyre. Personer med ren IT-bakgrunn vil måtte bruke betydelig mengder med tid for å lære seg disse programmene. Dette er unødvendig tidkrevende dersom en bare ønsker å utvikle veldig enkle grunnleggende kretser.

Det finnes programmer som er forenkle, det vil si at mulighetene er for begrenset til å kunne brukes i et undervisningsmiljø. Et eksempel på dette er «Simcir the circuit simulator», se figur 2. Programmet er en Applet som mangler muligheten til å åpne/lagre en fil. Vinduet er ikke stort nok til å tegne større kretser. «Macro tag» og «Flipflop» finnes heller ikke.

Digital Works (se figur 1) er et av de verktøyene mange universiteter og høgskoler bruker for tiden. Det enkle brukergrensesnittet til programmet gjør at det er godt egnet som et undervisningsverktøy. Digital Works er ikke et gratis program og koster 386 kr for en studentlisens og 5940 kr for en firmalisens. Men det finnes også en redusert gratis versjon som alle studenter kan laste ned og bruke på sin egen maskin. Digital Works kjører kun på Windows-installerte maskiner og siste oppdatering for programmet kom i 2001.



Figur 1: Digital Works



Figur 2: Simcir the circuit simulator version 1.2.1 http://www.d-project.com/simcir/

Digital Circuit

Krav til applikasjonen Digital Circuit er spesifisert med tanke på både studenter og undervisningsinstitusjoner.

1. Programmet skal være meget enkelt å bruke. Som nevnt tidligere er det mange ulike studenter som tar slike kurs og som ikke har dette som sin studieretning.

2. Programmet skal være plattformuavhengig. Dette sikrer at studenter og institusjoner som bruker forskjellige typer operativsystemer vil være i stand til å kjøre programmet. Dette innebærer også for eksempel at man kan lagre kretstegninger på en fil i ett operativsystem, og senere åpne denne filen på en hvilken som helst annen datamaskin uten å tenke på operativsystemet.

3. Programmet skal være et «Open Source»-prosjekt med «GNU (General Public License)» lisens. Kildekoden skal være åpen for alle. På denne måten kan alle som ønsker å laste ned kildekoden gjøre det og kan foreta endringer etter sitt behov så lenge det ikke strider med GNU-lisensen.

Programmeringsspråk

Java er valgt som programmeringsspråk i denne oppgaven fordi det ønskes at programmet skal være plattformuavhengig og i tillegg til det har jeg godt kjennskap til og kompentanse på Java.

1. JDK(Java Developement Kit), JRE (Java Runtime Environment) gjør Java-programmene plattformuavhengige og begge verktøyene er gratis.

2. Programmeringsspråket Java er godt utbredt i akademiske miljøer og mange universiteter og høgskoler tilbyr kurs i det.

Brukergrensesnittet

Brukergrensesnittet (GUI'en) er en sentral del i oppgaven siden mye av kommunikasjonen mellom bruker og programmet skal foregå gjennom skjerm, mus og tastatur. Det er viktig at brukergrensesnitt også er plattformuavhengig dersom selve programmet skal være det. Det er ikke mange GUI-programmer som er plattformuavhengige slik som Java. Men det finnes noen alternativer.

Tk er en API(Application programming interface) med basiselementer til implementering av plattformuavhengige brukergrensesnitt. Denne API'en er et åpent kildekodebibliotek med GNU-lisens og kan lastes ned fra http://www.tcl.tk.

TrollTech har også kommet med både en API og en designerverktøy som begge er plattformuavhengige.

Java har dette innebygd og det trengs derfor ingen eksterne verktøy. I Java har er det mulig å bruke to forskjellige type pakker, «AWT» og «Swing». Begge kan brukes til å implementere plattformuavhengige vindubaserte systemer, grafikk og brukergrensesnitt. I prosjektet vårt bruker vi begge pakkene. For mer om «AWT» og «Swing», se kapittel 2.1.2.

Prosjekt

I begynnelsen av prosjektet samarbeidet jeg med min medstudent Arne Gunnar Hoseth. Etter noen måneder ble vi enig om å jobbe hver for oss. Det eneste vi kom frem til sammen er å tegne porter med Java. Det vil derfor være noen likheter i et par kapittler, nærmere bestemt 4 og 5.

Del I

Bakgrunn

Kapittel 1

Digitale kretser

Ordet digital kommer av det engelske ordet DIGIT. Digitale signaler er signaler som bare har to verdier. PÅ eller AV. Dette betegnes med HØY for på og LAV for av, eller «1» og «0».

Ettersom kretsteknikken har utviklet seg har det kommet logiske kretser i IC-teknikk. IC står for «Internal Circuit» som betyr integrerte kretser, altså at mer eller mindre kompliserte elektroniske kretser ligger pakket inn i en og samme brikke. De mest kjente og populære IC'ene er mikroprosessor, mikrokontroller, statisk minne og andre digitale logiske kretser, som for eksempel bildebrikker i digitale kameraer.

En enkelt IC-brikke kan inneholde flere standardiserte koblinger. Det er kombinasjon av inngangsignaler som avgjør hvilket utgangssignal vi får fra koblingen. Disse koblingene består av det vi kaller for logiske elementer. En logisk krets kan bestå av komponenter som for eksempel ulike type porter, flipflop, vipper, motstander og dioder. Disse kombinerer vi slik at vi kan få den funksjonen som er nødvendig for å utføre en bestemt oppgave.

1.1 Ulike type porter

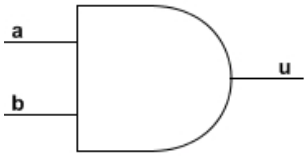
Digitale kretsene består av noen enkle grunnfunksjoner som danner basisen for alle logiske systemer. Disse kan kombineres på forskjellige måter for å oppnå de funksjonene vi ønsker. For eksempel kan en lage logiske systemer som implementerer funksjonalitet for styring av alarm, kraner eller kanskje sikringssystemene om bord på et jagerfly. Disse grunnfunksjonene er som listet under.

1. OG Eng. AND
2. ELLER Eng. OR
3. IKKE Eng. NOT
4. IKKE OG Eng. NAND

5. IKKE ELLER Eng. NOR
6. EKSKLUSIV ELLER Eng. XOR
7. IKKE EKSKLUSIV ELLER Eng. NXOR

I tillegg til disse grunnfunksjonene brukes det også vipper. Men siden disse ikke er implementert i Digital Circuit, blir de ikke beskrevet noe videre i oppgaven.

1.1.1 OG-port



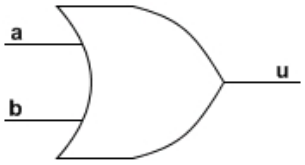
Figur 1.1: OG-port

a	b	u
0	0	0
1	0	0
0	1	0
1	1	1

Tabell 1.1: Sannhetstabellen til AND

Figur 1.1 viser en logisk OG-port hvor a og b er innganger og u er utgang. En logisk OG-port fungerer slik at en binær 1'er på begge innganer vil gi en binær 1'er på utgangen. Sannhetstabellen viser de resterende signalkombinasjonene en kan ha på inngangen og hvilken utgang disse resulterer i.

1.1.2 ELLER-port



Figur 1.2: ELLER-port

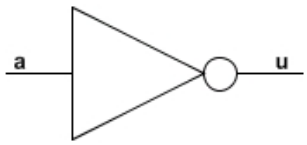
a	b	u
0	0	0
1	0	1
0	1	1
1	1	1

Tabell 1.2: Sannhetstabellen til OR

Figur 1.2 viser en logisk ELLER-port hvor a og b er innganger og u er utgangen. ELLER-porten virker slik at en 1'er på minst en av inngangene vil gi en 1'er på utgangen. Dersom begge inngangene får 0'ere vil utgangen også bli 0.

1.1.3 IKKE-port

Figuren 1.3 viser en logisk IKKE-port hvor den bare har en inngang a og en utgang u. Funksjonen til denne porten er slik at den sender det motsatte



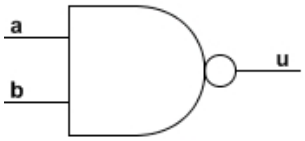
Figur 1.3: IKKE-port

a	u
0	1
1	0

Tabell 1.3: Sannhetstabellen til NOT

ut av hva den får inn. Dersom det kommer inn en 1'er på inngangen vil utgangen bli 0, og det motsatte hvis inngangen er en 0. Denne porten kalles også for en inverter.

1.1.4 IKKE OG-port



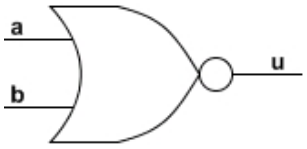
Figur 1.4: IKKE OG port

a	b	u
0	0	1
1	0	1
0	1	1
1	1	0

Tabell 1.4: Sannhetstabellen til NAND

Figur 1.4 viser bildet av en IKKE OG-port med to innganger a og b og en utgang u. IKKE OG port får vi også av en logisk sammensatt kobling hvor vi bruker en OG-port med en IKKE-port i serie. Funksjonmessig fungerer den motsatt av en AND-port. Sannhetstabellen viser dette. Her ser en at to 1'ere på innganen vil gi en 0'er på utgangen og 1 ellers.

1.1.5 IKKE ELLER-port



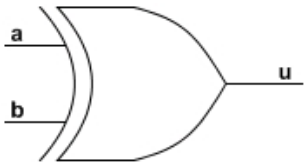
Figur 1.5: IKKE ELLER-port

a	b	u
0	0	1
1	0	0
0	1	0
1	1	0

Tabell 1.5: Sannhetstabellen til NOR

Figur 1.5 viser bildet av en IKKE ELLER-port med to innganger a og b og en utgang u. IKKE ELLER-port får vi også av en logisk sammensatt kobling hvor vi bruker en ELLER-port med en IKKE-port i serie. Denne porten er det motsatte av en ELLER-port. Dette kan ses ved å sammenlikne sannhetstabellene for en ELLER-port og en IKKE ELLER-port.

1.1.6 XOR-port



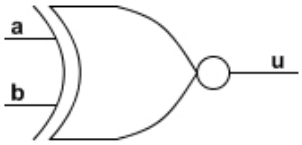
Figur 1.6: XOR-port

a	b	u
0	0	0
1	0	1
0	1	1
1	1	0

Tabell 1.6: Sannhetstabellen til XOR

Figur 1.6 viser bildet av en XOR-port med to innganger a og b og en utgang u. XOR-porten kan brukes til å utføre logiske addisjoner.

1.1.7 XNOR-port



Figur 1.7: XNOR-port

a	b	u
0	0	1
1	0	0
0	1	0
1	1	1

Tabell 1.7: Sannhetstabellen til XNOR

Figur 1.7 viser bildet av en XNOR-port med to innganger a og b og en utgang u. XNOR-port er det motsatte av en XOR-port funksjonmessig.

1.2 Eksempler på kretser vist med Digital Circuit

1.2.1 NAND-porter til å konstruere en XOR-port

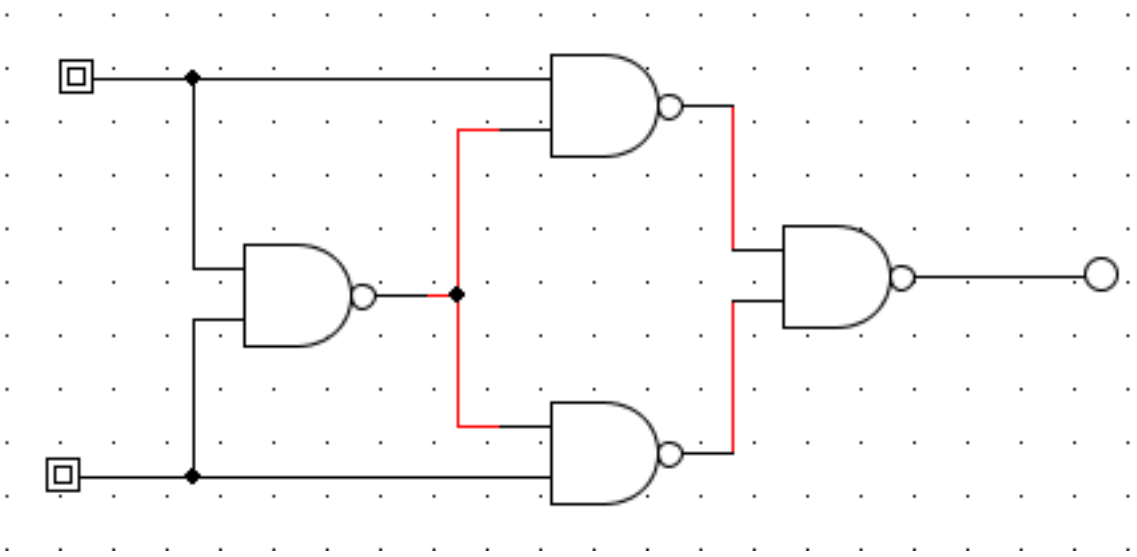
Kretsen vist på figur 1.8 er tegnet ved hjelp av Digital Circuit og de røde ledningene viser at signalet er 1.

1.2.2 NOR-porter til å konstruere en XOR port

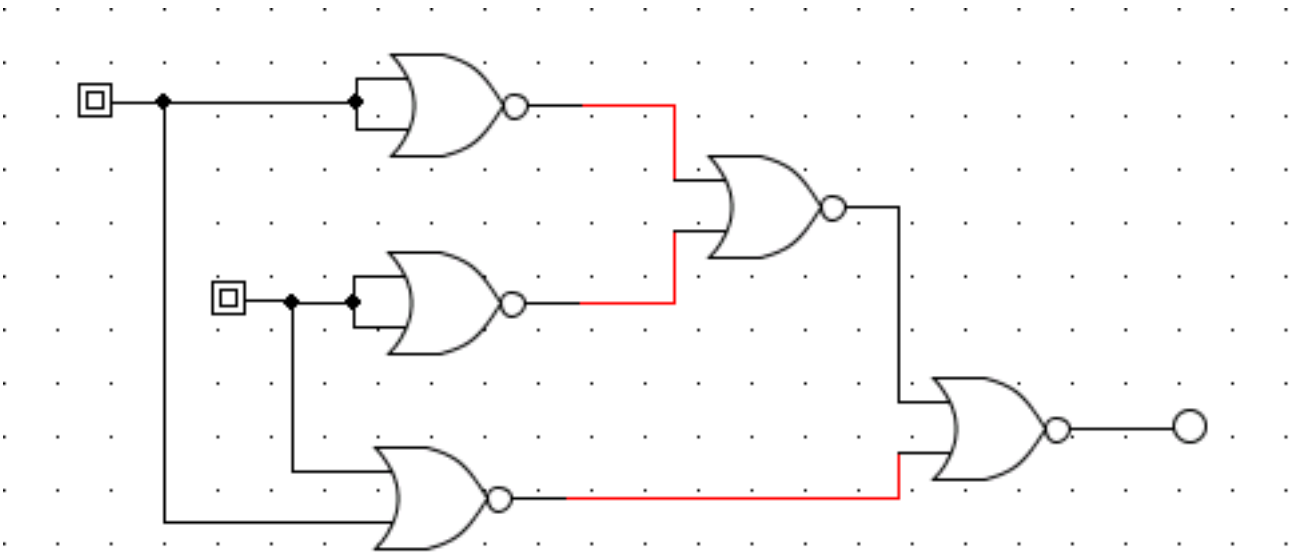
Kretsen vist på figur 1.9 er tegnet ved hjelp av Digital Circuit, legg merke til at ledningene endrer farge til rødt når signalet er 1.

1.2.3 Addere

En av de mest fundamentale funksjoner en datamaskin må utføre er å addere/subtrahere tall. Et adder kan vi bygge for mange typer tallsystemer, slik som BCD (Binary-coded decimal) eller excess-3, men de fleste addere opererer på binære tall på 2-er komplement form.



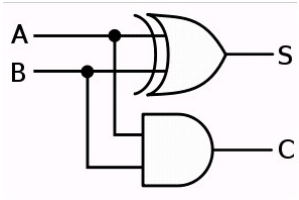
Figur 1.8: XOR-port konstruert av NAND-porter



Figur 1.9: XOR-port konstruert av NOR-porter

Det finnes to generelle typer en bits-adder. Den første heter halv-adder og den andre heter full-adder. Figur 1.10 viser en halv-adder hvor A og B er innganger og S er summen og C er mente (carry).

.



Figur 1.10: Halv-adder[7]

Inngang		Utgang	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Tabell 1.8: Halv-adder tabell

Figuren 1.10 viser en krets som gjør bare halve jobben av en addisjon fordi den mangler mente inn. Vi kaller derfor en slik krets for en halv-adder. For å skape en krets som gjør hele jobbe med mente inn og ut, kan vi sette sammen to halv-addere. Figur 1.11 viser en full-adder med sannhetstabellen 1.9.

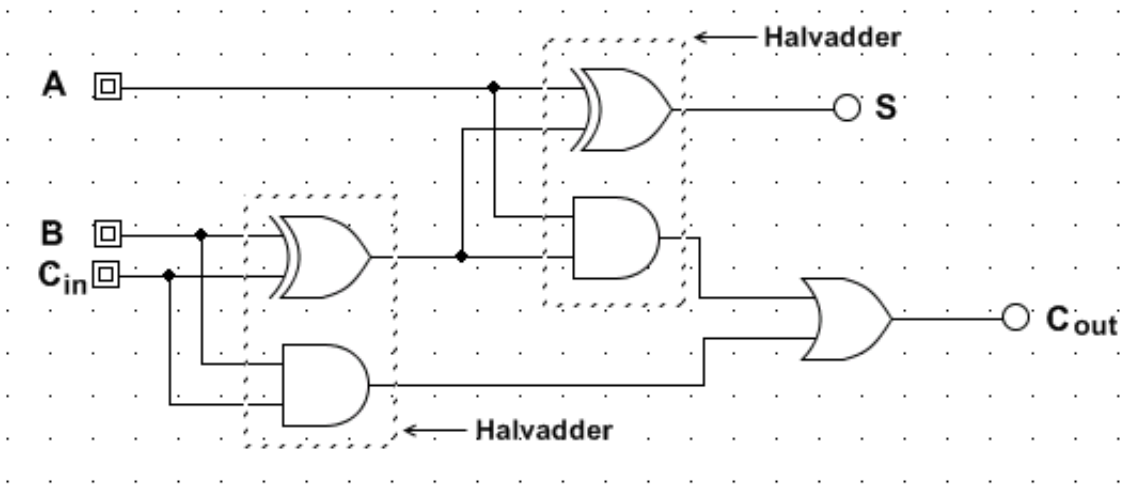
Inngang			Utgang	
A	B	C-in	C-out	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Tabell 1.9: Full-adder tabell

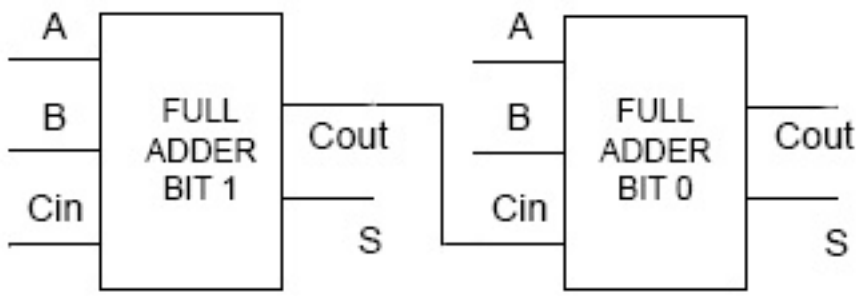
Med sannhetstabellen 1.9 ser vi at en full adder kan utføre en fullstendig addisjon. Men for å kunne addere binære tall med flere bit må vi koble sammen flere fulladdere, se figur 1.12.

1.3 Full-adder tegning og simulering med Digital Circuit

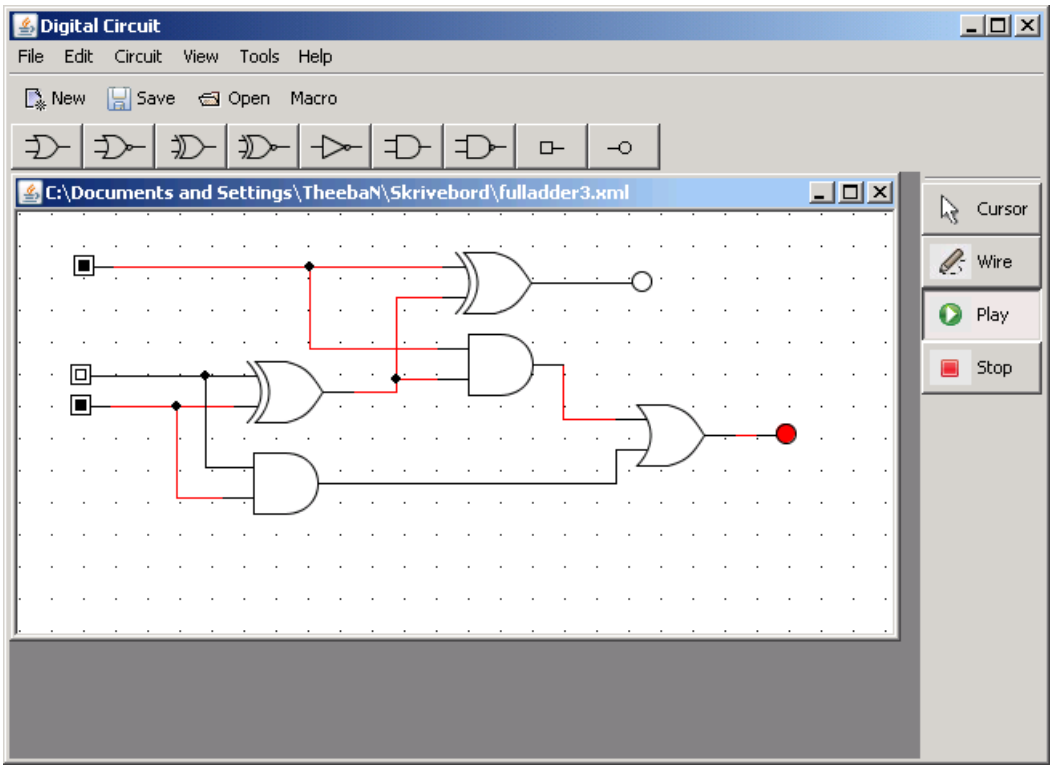
Figuren 1.13 viser et fullstendig bildet av en full-adder i aksjon laget med Digital Circuit.



Figur 1.11: Full adder



Figur 1.12: To full-addere koblet sammen



Figur 1.13: Konstruksjon og simulering av full-adder med Digital Circuit

Kapittel 2

Tråder

Tråder kan benyttes til å simulere en krets i Digital Circuit applikasjonen. Med tråder kan man samkjøre flere sekvenser av en oppgave, som kan utføres systematisk eller uavhengige av hverandre. Dette kapittelet gir en overordnetbeskrivelse av hva tråder og prosesser er, og en beskrivelse av hvordan de implementeres ved hjelp av java.

2.1 Bakgrunn

Tråder er en fornorsking av det engelske ordet threads som er en forkortelse av uttrykket «thread of execution». Tråder er en sekvens av instruksjoner som kan samkjøres parallelt med andre tråder i samme prosess. En annen definisjon for tråder er at de er deler av et program som kan kjøres uavhengig og samtidig med andre deler av programmet.

Videre i kapittelet skal vi blant annet se på parallellitet og gå nærmere inn på hva en prosess er kontra en tråd.

2.2 Parallellitet

2.2.1 Prosesser og tråder

En metafor for sammenhengen mellom to uavhengige prosesser og tråder er to verksteder med to mekanikere. I en prosess vil de reparere hver sin bil, mens for tråder ville delt ressursene og reparert to biler i et slags fellesverksted.

Parallellprogrammering er en måte å løse en oppgave ved å kjøre flere samarbeidende programmer eller programbiter samtidig. Selv om en tråd har kommet til et punkt hvor den må stoppe, kan de andre trådene holde på med sin oppgave[42]. Oppgaver som kan løses ved parallellitet kan være numeriske algoritmer og programmer med en høy grad av parallellitet som for eksempel matrise-multiplikasjoner. Et annet eksempel er brukerinteraksjon. Hvis en tråd venter på en input fra brukeren, for eksempel «C» for å avbryte

jobben, kan en annen tråd jobbe med sin oppgave, for eksempel med skrive en rapport.

De fleste maskiner har større kapasitet enn det et enkelt program trenger og ved bruk av parallelle programmer kan effektiviteten økes dramatisk. Da kan en bruker kjøre flere programmer samtidig. Dette er også en prosess for systemer med flere brukere som trenger å jobbe med samme data samtidig. Det finnes to varianter for parallellprogrammering, prosesser og tråder. I samarbeidende prosesser sender prosessene meldinger til hverandre eller leser og skriver i felles minneområde men ikke samtidig.

2.2.2 Ekte parallellitet

Det finnes to typer trådmodeller: singlethreaded og multithreaded. Single-threaded prosesser er når en prosess lager en tråd og multi-threaded prosess er når en prosess lager flere tråder. Den flertrådede prosessen vil da være det som er aktuelt for parallellitet.

Ekte parallellitet kan være når man har en maskin med flere CPUer for eksempel «Dual Core» med to mikroprosessorer og en tråd tildelt i hver av dem. Er det tildelt flere tråder enn det er prosessorer kalles dette pseudo-parallellitet, og dette finner man i flere OSer der systemet veksler mellom flere prosesser avhengig av time slice for å simulere ekte parallellitet. Time slice er intervallet av tid som en prosess er tillatt å kjøre i et preemptivt multitasking system. Ekte parallellitet kan man også få ved å kjøre på flere maskiner over et nettverk.

Hensikten med ekte parallellitet er at hvis man kjører på flere prosessorer, kan deler av programmet bli kjørt samtidig. Oppgaven blir dermed fortere ferdig enn hvis bare én prosessor skulle ha kjørt hele programmet. Strukturen i programmet blir også bedre med multi-threads. Programmer kan deles opp i flere uavhengige enheter av utførelser istedenfor som en enkelt monolittisk tråd.

Ulempen med multi-threads er at de er vanskeligere å debugge og programmere på en slik måte at kode unngår race-conditions og vranglås. Vranglås kan forekomme når to eller flere tråder venter på hverandre uten å gjøre noe som helst. Dette fører videre til at systemet henger seg. Tråder er vanskelige å synkronisere og man må koordinere tilgangen til felles data med låser for ellers kan data bli korrumpert.

Siden tråder krever færre OS ressurser er de raskere å lage og fjerne. Dette er en av grunnene til at de bør brukes til nye applikasjoner.

2.3 Prosesser

En prosess er uavhengig, har masse tilstandsinformasjon, separate minne-adresser og interne prosess-kommunikasjonsmekanismer. Vi skal her nøye oss med å snakke om prosesser i Unix. I Unix bruker man fork-metoden til å lage

en barneprosess. Dette er en kopi av den originale prosessen som også kalles for forelder-prosessen. PIDen til den nye barneprosessen vil bli returnert av fork-metoden til forelderprosessen. I barne prosessen vil retur verdien til fork-metoden være 0.

Herunder følger et eksempel i Perl.

```
1  #! /usr/bin/perl
2  $pid = fork();  # fork() starter en ny prosess, en eksakt kopi
3
4  print "hei\n";  # Utføres av begge
5
6  if($pid == 0)
7       {
8        # Utføres kun av child-prosessen
9          print "Jeg er barne-prosessen.\n";
10      }
11 else
12      {
13       # Utføres kun av parent-prosessen
14         print "Jeg er foreldre prosessen.\n";
15      }
```

```
Kjøreeksempel:
hei
Jeg er barne-prosessen.
hei
Jeg er foreldre prosessen.
```

Først invokerer man fork funksjonen og får returnert et heltall. Deretter tar man en sjekk om at hvis PIDen er 0 da er det barne prosessen men ellers er det foreldre prosessen.

Et eksempel i C for bytting av prosesser, se side 18.

2.4 Tråder

Den vesentligste forskjellen mellom tråder og prosesser er at tråder deler felles minne. Tråder er uavhengige av hverandre og de deler ressurser, minne og midlertidig informasjon fra en enkel prosess. Ved sammenhengssvitsjing det vil si context switch skjer dette fortere med tråder enn med prosesser. Context switch er når et operativsystem switcher mellom prosesser[19]. Et eksempel i Java for bytting av tråder, se side 19.

Vi ser her at eksempelet med trådbytting tok betydelig mindre tid enn eksempelet med bytting av prosesser.

Tråder kan dele minne og har bra ytelse for eksempel raskere context switch. Den store fordelen med multi-threaded applikasjoner er at de deler CPUer og dermed blir de kjørt parallelt. Applikasjonen går raskere enn en

```c
#include <signal.h>
#include <stdio.h>
#include <sys/times.h>
#include <sys/types.h>
#include <unistd.h>
#include <limits.h>

#define N 500000

float cputime(void)
{
  struct tms t;

  times(&t);
  return (float)(t.tms_utime+t.tms_stime+
                 t.tms_cutime+t.tms_cstime)
         /sysconf(_SC_CLK_TCK);
}

int main(void)
{
  int   pipd1[2], pipd2[2], i;
  char  buf = 'x';
  pid_t barne_pid;

  pipe(pipd1);  pipe(pipd2);

  barne_pid = fork();
  if (barne_pid > 0) {
    /* Opphavet: */

    float start_tid = cputime(), brukt_tid;

    for (i = 1;  i < N;  ++i) {
      write(pipd1[1], &buf, 1);  read (pipd2[0], &buf, 1);
    }
    brukt_tid = cputime()-start_tid;
    kill(barne_pid, SIGTERM);

    printf("%d prosessbytter tok %.2fs, dvs %.2fµs/bytte.\n",
      2*N, brukt_tid, brukt_tid/(2.0*N)*1000000.0);
  } else {
    /* Barnet: */

    while (1) {
      read (pipd1[0], &buf, 1);  write(pipd2[1], &buf, 1);
    }
  }
}
```

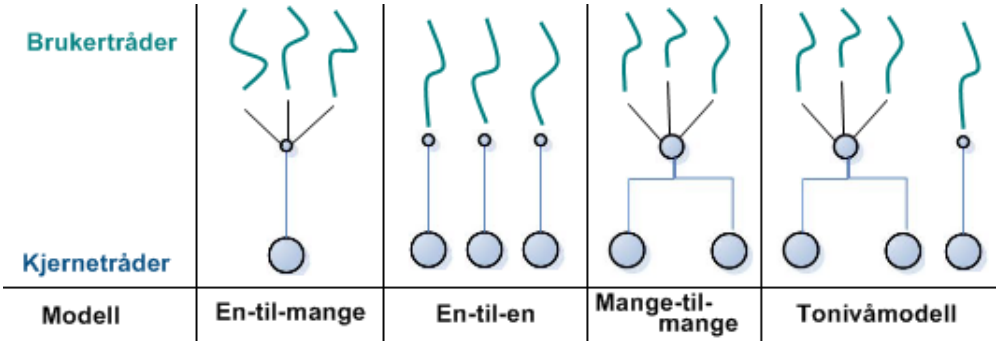
Nedenfor følger output av programmet:

```
1000000 prosessbytter tok 4.34s, dvs 4.34µs/bytte.
```

```
1  import java.text.*;
2
3  public class ThreadSwitch extends Thread {
4
5      String[] array1 = new String[2];
6      String[] array2 = new String[2];
7      String[] buf = {"Testing Testing Testing Testing Testing Testing
8  Testing"};
9
10     int N = 1000000;
11     DecimalFormat df = new DecimalFormat("0.00");
12
13     public ThreadSwitch(String name){
14         super(name);
15     }
16
17     public void run() {
18         long time = System.currentTimeMillis();
19         for (int i = 0; i < N; i++)
20         {
21             this.yield();
22         }
23         long time2 = System.currentTimeMillis();
24         long timeused = (time2 - time);
25         float antall_sec = timeused / (float)1000;
26
27         System.out.println( N + " trådbytter tok " +
28 df.format(antall_sec) + "s.");
29
30     }
31
32     public static void main(String args[]) {
33
34         Thread th = new ThreadSwitch("th");
35         Thread th2 = new ThreadSwitch("th2");
36         th.start();
37         th2.start();
38
39     }
40
41 }
```

Nedenfor ser vi output fra programmet:

```
1000000 trådbytter tok 0,61s.
1000000 trådbytter tok 0,61s.
```



Figur 2.1: Trådmodell

singlethreaded applikasjon som kun får tildelt en CPU. Applikasjonen er også raskere til å respondere. Hvis en bruker av en multi-threaded GUI har satt i gang en aktivitet trenger ikke brukeren å vente til aktiviteten blir ferdig før en ny en kan startes.

Når det gjelder kommunikasjon og deling av ressurser er det enklere med tråder enn med prosesser. Økonomisk sett er tråder mye bedre enn prosesser fordi den bruker mindre minne, trenger ikke egne dataområder og kan fordele seg til flere CPUer hvis maskin har det for å behandle oppgaver. Tråder kan kjøres uavhengige av hverandre i motsetning til prosesser men det kommer an på hvordan de er implementert. Selv om en tråd venter på input fra brukeren eller henger seg vil prosessen kjøre videre.

2.5 Tråder i operativsystemer

Det finnes tre hovedmodeller for tråder i OSkjernen som er en-til-en, en-til-mange, mange-til-mange og tonivåmodell. Figur 2.1 illustrerer fordelingen av kjerne-tråder som befinner seg inne i kjernen til OSet. Bruker-tråder er tråder som blir håndtert av brukerapplikasjoner.

2.5.1 En-til-mange

Flere brukertråder blir håndtert av kun en kjernetråd. Denne modellen blir brukt i systemer som ikke støtter flere kjernetråder. I de systemene som har denne type modell skjer det ingen ekte parallellitet. Hver tråd i denne modellen er schedulert som en prosess. «Scheduling» er når man deler CPUen mellom flere prosesser. Java «green threads» er et eksempel på hvor JVM schedulerer selv og ingen multi-tasking[1].

2.5.2 En-til-en

Hver enkel brukertråd i denne modellen blir håndtert av hver enkel kjernetråd. Derfor kan trådene i denne modellen kjøres parallellt, og trådene scheduleres uavhengig av hverandre. Eksampler på operativsystemer som bruker denne type modell er Windows 95/98/NT/2000/XP og OS/2[1].

2.5.3 Mange-til-mange

Mange brukertråder i denne modellen kan håndteres av like mange eller mindre antall kjernetråder. Operativsystemet støtter for å opprette tilstrekkelige antall kjernetråder for å håndtere disse brukertrådene. Samtidighet i denne modellen er mindre enn En-til-en modell, men bedre enn En-til-mange. Det finnes ulike Operativsystemer som benytter seg av de ulike modellene. Eksempler på operativsystemer som bruker denne modellen er Solaris 2, Windows NT/2000[1].

2.5.4 Tonivåmodell

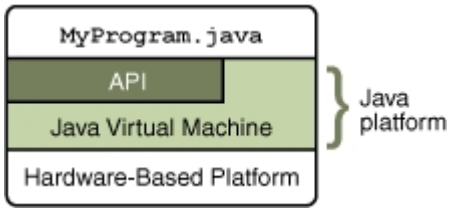
Denne modellen er gangske lik den Mange-til-mange modellen, men denne modellen lar i tilleggg en brukertråd til å bli knyttet til en kjernetråd. Operativsystemer som bruker denne modellen er IRIX, HP-UX, Tru64 UNIX og Solaris 8 og tidligere versjoner[1].

2.5.5 Håndtering av tråder i operativsystemer

Tråder på brukernivå er håndtert av applikasjoner ved å bruke et trådbibliotek. Kjernen vet ikke om trådene men håndterer fortsatt prosessaktivitet. Bytting av tråder krever ikke privilegier fra kjernen og man slipper dermed å skifte modus. Schedulering er avhengig av applikasjonen det vil si den er spesifikk for applikasjonen og kan da velge den beste algoritmen.

Hvis en tråd utfører et systemkall vil hele prosessen bli blokkert unntatt trådbiblioteket hvor tråden fortsatt kjøres. Tilstandene til tråder er da uavhengig av tilstandene til prosesser. Tråder på bruker nivå kan kjøres på et hvilket som helst operativsystem siden den trenger bare et trådbibliotek. Ulemper med tråder på brukernivå er at disse ved de fleste systemkallene for blokkering, også blokkerer hele prosessen. Dermed blir alle trådene i prosessen også blokkert. En annen ulempe er at kjernen kan bare tilegne prosesser til prosessorer, så to tråder på samme prosess kan ikke kjøres samtidig på to prosessorer.

Tråder på kjernenivå er håndtert av kjernen og ikke noe trådbibliotek men en API det vil si systemkall til kjerne trådfasiliteten eksisterer. Kjernen bevarer informasjon for prosessen og for trådene og bytting mellom trådene krever at schedulering av kjernen er utført på en tråd basis. Fordeler med tråder på kjerne nivå er at kjernen kan schedulere mange tråder fra den



Figur 2.2: JVM[10]

samme prosessen på mange prosessorer samtidig og blokkering er gjort på tråd nivå. Men bytting av tråder innenfor den samme prosessen involverer kjernen. For eksempel hvis vi har to byttinger av moduser per bytting av tråd vil dette medføre i en signifikant treghet.

Windows NT har både prosesser og tråder. På Unix er hver prosess nesten lineært- ikke parallelt- men sekvensielt.

2.6 Implementasjon

Java velges som programmeringsspråk for implementering av tråder. Videre står det beskrevet litt om Java og dets egenskaper.

Java har også støtte for multitreading i språket. Tråd i Java er instans av klassen Thread og man kan opprette og slette som andre objekter[21].

2.6.1 JVM

Java er både en plattform og et programmeringsspråk. Denne plattformen består av to deler.

- JVM - Java Virtual Machine.
- Java API - Java Application Programming Interface.

JVM er Javas virtuelle maskin og er bestående av et sett med programmer og datastrukturer. Den virtuelle maskinen er et lag som ligger mellom maskinen og Java-programmet som blir kjørt. Dette laget kan finnes i alle maskiner og dette medfører at programmet kan eksekveres på ulike maskiner med ulik plattform. Java API er biblioteket bestående av ett sett med ferdiglagde programvare komponenter.

Selv om man kjører flere tråder samtidig vil ikke tiden det tar for å løse en oppgave bli redusert. Grunnen til dette er at JVM ikke kan distribuere tråder gjennom flere CPUer og dermed kan ikke dette nyttegjøres på PCer med flere CPUer. På Windows går det an å kjøre på flere CPUer fra JVM, men fungerer ikke på Unix[40].

Hvor mye hver tråd inneholder av virtuelt minne allokert for stakken avhenger av hva slags prosessor maskinen har. Nedenfor vil dere finne en

tabell som forklarer dette. Hvis man hadde hatt tusenvis av tråder ville dette ha sløst signifikant mengde av størrelse av stakk område. Det minimum av størrelse for 1.3 og 1.4 er 64k og i 1.2 er 32k som man kan endre via Xss flagget[18].

- Prosessorer - Størrelse i kB
- Sparc - 512
- Intel 1.3 32-bit VM - 256
- Intel 1.4 32-bit VM - 256
- Sparc 64-bit 1.4 VM - 1024
- Sparc 64-bit 1.2 VM - 128

2.6.2 Tråder, grønne og ekte

Java støtter to typer tråder kalt grønne tråder og ekte tråder[5]. I Java green threads scheduleres av JVM istedenfor operativ systemet. Grønne tråder etterlikner «multi-threaded» miljøer uten å bruke noen av OS sine egenskaper. Siden den ikke er OS-avhengig bruker den ikke den OS-området («kernel space») og dermed bruker den bruker-området («user space»).

Siden den er opprettet i bruker nivå vil grønne tråder være lettere enn ekte tråder. Grønne tråder følger prinsippet til Java ved at trådene kan kjøres på alle plattformer selv om maskinen ikke har kooperativ multitasking. Ulempen med grønne tråder er at de ikke kan ta full nytte gjennom av et system med flere CPUer på Unix og de fleste andre operativsystemer. Dette er fordi disse trådene ikke er på samme nivå som operativsystemet. Men på Windows er det støtte for dette. En annen ulempe er at ekte tråder er raskere enn grønne tråder.

I grønne tråder er det bare en operativsystemtråd som blir brukt av JVM for alle Java tråder du bruker i dine programmer, mens i ekte tråder er det slik at for hver tråd du skaper i Java får du en separat operativsystemtråd.

På Windows NT finnes bare ekte tråder, mens på Solaris plattform, finnes begge modeller og på eldre JVM på Solaris finnes bare grønne tråder.

2.6.2.1 Tråder i Java

Det er tre forskjellige måter å ta i bruk tråder i Java.

1. Implementere Runnable klassen
2. Arver Thread klassen
3. Anonym klasse

2.6.2.2 Runnable-klassen

Første måten er at man bruker Runnable-grensesnitt slik som vist herunder:

```
1  import java.io.*;
2  import java.lang.Thread;
3  class X implements Runnable {
4                  X() {
5                      Thread t = new Thread(this);
6                      t.start();
7                  }
8                  public void run() {
9                      Work();
10                 }
11                 public void Work() {
12                 }
13  }
```

2.6.2.3 Thread-klassen

Den andre måten å implementere tråd på er ved å arve Thread-klassen som vist her under. Her er tråd definert som en separat klasse.

```
1  import java.io.*;
2  import java.lang.Thread;
3
4  class Y extends Thread {
5                  Y() {
6                      start();
7                  }
8                  public void run() {
9                      Work();
10                 }
11                 public void Work() {
12                 }
13 }
```

2.6.2.4 Anonym-klasse

Den siste måten er en anonym metode. Det vil si man kan implementere Thread akkurat der man vil ved å bruke Adapter teknikk metoden i Java, se side 25.

2.6.2.5 Komplett eksempel

For et komplett eksempel, se side 25.

2.6.2.6 Metoder i Thread-klassen

- start() - Denne metoden allokerer minne, stack osv. og kaller på run.

..

Anonym-klasse:

```
public class AApp {
     public static void main( String args[] ) {
               Thread t = new Thread (new Runnable() {
                                                    public void run() {
                                                         Work();
                                                    } } );
               t.start();
          }
     public static void Work() {
     }
   }
```

Komplett eksempel:

```
class SimpleThread extends Thread {

    public SimpleThread(String name) {
        super(name);
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + getName());
            try {
                //sover i tilfeldig antall sekunder
                sleep((int)(Math.random() * 1000));
            } catch (InterruptedException e) {}
        }
        System.out.println("DONE! " + getName());
    }

    //Tester to tråder
    public static void main (String args[]) {
        new SimpleThread("Thread1").start();
        new SimpleThread("Thread2").start();
    }

}
```

- run() - Denne metoden utfører jobben som tråden skal utføre.
- yield() - Når denne metoden blir kalt gir den CPUen fra seg.
- setPriority() - Metoden brukes til å sette trådens prioritet fra 1 til 10 og standard prioritet er 5.
- sleep(ms) - Du kan kalle opp denne metoden ved å gi antall millisekunder tråden skal sove.

2.6.2.7 Ekte tråder

Native threads er ekte tråder og finnes på kjerne området (kernel space). Ekte tråder er tråder som blir styrt direkte fra et operativsystem i motsetning til green threads som blir styrt av en virtuell maskin. Det er forutsatt at operativsystemet støtter tråder. Operativsystemet kan schedulere tråder gjennom flere CPUer, men da kan den også bruke den scheduleringss-algoritmen den ønsker og dermed ta kontroll vekk fra utvikleren.

2.6.3 Sikkerhet, syncronized

Noen datastrukturer er trådsikre, mens andre ikke. Eksempler på trådsikre strukturer er Vector, HashTable, mens ArrayList og HashMap er usikre. Kode som er trådsikkert er tregere enn kode som ikke er det[20]. Dette medfører at du bør tenke deg om hvis det er nødvendig med trådsikker kode. Når vi snakker om GUI-bibliotekene i Java er AWT trådsikker, mens ikke Swing.

2.6.3.1 Nøkkelordet syncronized

Synkronisering brukes når vi trenger å behandle en ressurs om gangen. Det vil si at bare en tråd om gangen får lov til å aksesserer for eksempel en felles variabel eller kodeblokk. Man vil unngå at to prosesser endrer felles data samtidig. Det vil blant annet si at en prosess ikke bør lese felles data mens en annen prosess endrer den. Prosessen må da vente på den andre prosessen. Det at en prosess jobber en av gangen på felles data kalles serialisering og da unngår man race condition som er en konkurranse mellom prosesser om å komme inn i et kritisk avsnitt. Dette er viktig fordi en vil unngå at for eksempel to tråder er innenfor den samme ressursen samtidig for da kan verdien til resursen bli feil. Brukeren kan selv serialisere sine prosesser da mulighetene til dette ligger i OSet. I Java kan du synkronisere ved å bruke nøkkelordet synchronized.

Java har en variant av monitorkonseptet der alle objekter kan fungere som monitorer. Man kan indikere hvilken kodeblokk man vil beskytte og hvilket objekt med dets monitor som skal få tilgang til denne koden. Det finnes

to måter å beskytte kode med en monitor på, og begge bruker nøkkelordet synchronized[43].

Den første måten er å sette synchronized i deklareringen til en metode og dermed beskyttes av monitoren til objektet metoden kjøres på. Dersom alle metoder i en klasse er synchronized kalles dette for en klassisk monitor. Her blir all koden til metoden kalt for kritisk avsnitt.

```
1 class Foo {
2  synchronized static void bar() {
3    ...
4  }
5 }
```

Den andre måten å beskytte er ved å lage en kodeblokk som er synkronisert. Man kan sette synchronized inne i metoden slik som vist under. Da blir koden som ligger innenfor nøkkelordet synchronized kalt for kritisk avsnitt. Dette kan man gjøre på følgende måte.

```
1 void bar() {
2  synchronized (this) {
3    ...
4  }
5 }
```

Kort sagt kan man forklare prosessen når vi setter synchronized:

- Maks en tråd skal kommer inn i noen av de synkroniserte metodene eller kodeblokk i dette objektet samtidig. Dette vil da være monitorens eier.
- En tråd om gangen slippes til, og da må de andre trådene vente.
- Når den ene tråden er ferdig, slippes de som venter, men en etter en.

2.7 Historie

Tråder stammer helt tilbake fra 1965 med Berkeley Timesharing System[16]. Disse ble ikke kalt for tråder men prosesser. Rundt 1970 lagde Max Smith en implementasjon av prosesser på Multics. Denne implementasjonen brukte flere stakker i en enkelt tungvektsprosess til å støtte bakgrunnskompileringer. Mens Multics ble utviklet lagde IBM et språk kalt PL/I som er en viktig utvikling til tråder.

Unix kom tidlig i 1970-årene. Notasjonen til Unix for en prosess er at det er en sekvensiell tråd av kontroll samt et virtuelt minneområde. Siden hver prosess ikke deler minneområde interagerer de gjennom pipes, signaler og så videre. Etter en stund begynte brukerne av Unix å savne de gamle prosessene som kunne dele minne. Dette førte til oppfinnelsen av tråder som delte minneadressen til en enkelt Unix prosess. Tråder har også vært lenge brukt i telekommunikasjonsapplikasjoner.

2.8 Framtiden

I framtiden vil JDK på Solaris støtte ekte tråder[17]. Det vil komme prosessor med flere prosessorer enn to eller fire som vi har i dag. Det kan komme prosessorer med 1024 mikroprosessorer på ett chipsett. Da kan for eksempel JVM kreere en tråd til hver mikroprosessor og forbedre ytelsen betraktelig.

2.9 Konklusjon

Tråder deler felles minne mens prosesser gjør ikke det. Ett enkelt tråd kalles single-threaded og flere tråder sammen kalles multi-threaded. Det er en økende tendens til å øke antallet mikroprosessorer i et chipsett slik at programmer som er multi-threaded kan kjøres raskere. Multi-threadede programmer er også vanskelige å kode og debugge. Tråder i operativsystemer er enten en-til-mange, en-til-en, mange-til-mange eller tonivåmodell.

Kapittel 3

GUI I Java

Java er både en plattform og et programmeringsspråk som har vært under kontinuerlig utvikling siden 1995. I dag tilbyr Java en rekke gode byggeklasser som egner seg godt til prosjekter som denne. Java har tusenvis av klasser under sin standard klassebibliotek. Det er ikke lett å kjenne til alle disse klassene selv om man har arbeidet lenge med Java. I dette kapittelet vil vi derfor presentere noen av de viktigste Swing-klassene som trengs for å implementere funksjonaliteten og GUI.

3.1 Ulike GUI-pakker

Swing og AWT tilbyr et grafisk brukergrensesnitt for Java programmer. Swing er en del av Sun Microsystems sin JFC som står for Java Foundation Classes. Disse klassene utgjør et sett av GUI komponenter og tjenester som dramatisk forenkler utviklingen og gir deg samme kvalitet som kommersielle skrivebord og Internet/intranett applikasjoner. GUI delen har en sentral rolle og skal utgjøre en stor del av denne oppgaven. Derfor har vi tenkt å se litt nærmere på mulighetene vi har med Swing og de elementene i Java som er viktig for vår oppgave.

3.1.1 AWT

AWT står for The Abstract Windowing Toolkit og gir grunnleggende muligheter for å skape et grafisk brukergrensesnitt og til å tegne grafikk. Denne har vært en del av kjernen til Java siden Java 1.0. AWT er bygd som et ekstra lag over systemets innebygde GUI for den underliggende plattformen, og er derfor avhengig av operativsystemets GUI-system. Dette innebærer at dersom vi lager et grafisk trykknapp med AWT, vil den lage et trykknapp som tilhører den plattformen vi kjører applikasjonen på. I versjon Java 1.1, er AWT utvidet til å lage lettvektige GUI komponenter[36].

Figur 3.1: AWT og Swing-knapper

Figur 3.1 viser et bilde av en Java applikasjon med en AWT og en Swing-knapp. Disse er kjørt under flere operativsystemer.

Som vi ser er AWT knappene ulike på de forskjellige operativsystemer mens Swing knappene beholder den samme utseende.

Koden til applikasjonen:

```
11 public class AWTTest extends JFrame{
12
13   public AWTTest()
14   {
15     try {
16           UIManager.setLookAndFeel(
17               UIManager.getCrossPlatformLookAndFeelClassName());
18         } catch (Exception e) {
19         }
20
21     this.setTitle("AWT og Swing - Test");
22     Button but = new Button("awt.Button");
23     JButton but2 = new JButton("swing.JButton");
24
25
26     this.setLayout(new FlowLayout());
27     this.add(but);
28     this.add(but2);
29     this.setBackground(Color.WHITE);
30     this.setSize(200,100);
31     this.setVisible(true);
32   }
33
34   public static void main(String args[])
35   {
36     AWTTest awtframe = new AWTTest();
37     awtframe.addWindowListener(new WindowAdapter(){
38           public void windowClosing(WindowEvent we){
39             System.exit(0);
40           }
41         });
42
43   }
44 }
```

3.1.1.1 Graphics2D

Graphics2D er en del av APIet til Java 2D fra Sun. Denne pakken er en utvidelse av grafikklassen Graphics som gir deg mulighet til å effektivisere grafikk. Ved å konvertere Graphics objekt til Graphics2D kan vi enkelt bruke metodene som er tilgjengelig i denne klassen til for eksempel å tegne standard former og til å transformere dem.

I første omgang har vi kun brukt Graphichs2D til å tegne de utvalgte portene. Dette har blitt gjort ved å sette sammen instanser av klassene Line2D.Double, Ellipse2D.Double, QuadCurve2D.Double og CubicCurve2D.Double. Disse klassene ligger i `java.awt.geom` pakken. Bruker man slike klasser for å tegne, må man også bruke dens indre klasse av typen Double eller Float. Dette gjøres for å spesifisere presisjonen til punktene.

3.1.2 Swing

Swing er et forbedret GUI verktøy og den er tilgjengelig som en del av kjernen til Java 2 plattformen. GUI verktøyet er en utvidet bibliotek basert på AWT. Swing inkluderer nye og forbedrete komponenter som gir en bedre utseendet til GUI'en. AWT er det grunnleggende grafiske verktøyet men ved å bruke Swing istedenfor AWT har du også mer funksjonalitet tilgjengelig.

Swing er fullstendig implementert i Java språket og har vært tilgjengelig siden JDK 1.1 og er en del av rammeverket til lettvekts brukergrensesnittet. Selv om Swing har vært tilgjengelig siden JDK 1.1 har funksjoner for Java2D og noen lyd formater ikke blitt støttet før JDK 1.2[37].

Alle GUI komponenter som er gitt av Swing er lettvektskomponenter. Swing gjør det også lett å skrive en grafisk applikasjon som ser og oppfører seg likt på alle operativsystemer. Hvordan et grafisk Swing trykknapp ser ut på noen av de utvalgte operativsystemer ser dere på figur 3.1.

3.1.2.1 Hvordan bruker man Swing?

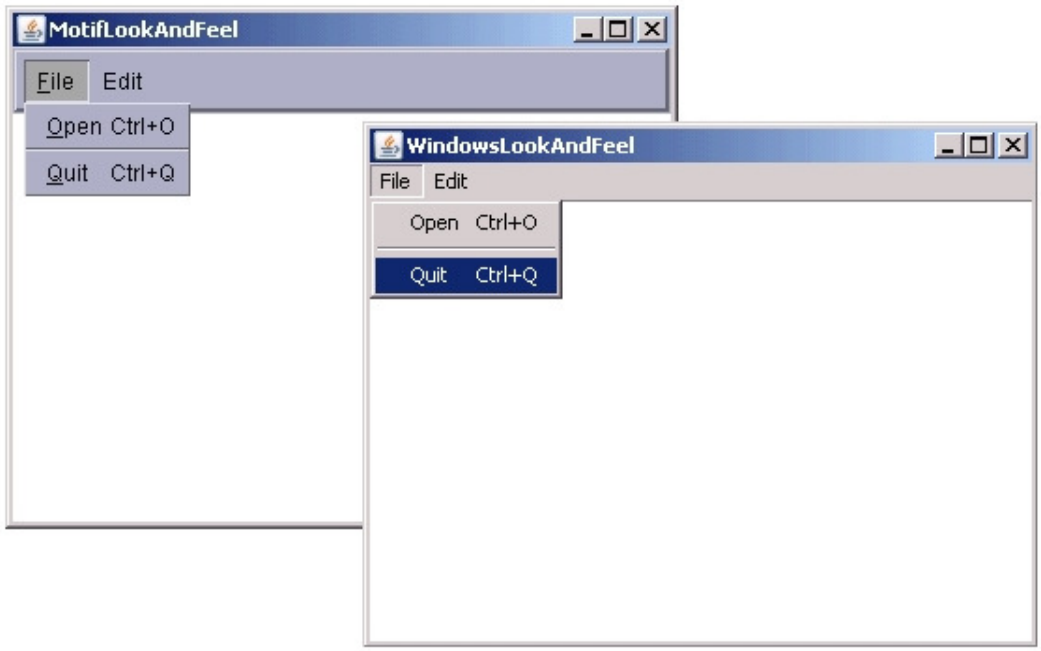
Swing komponenter inkluderer alt fra knapper, splitting av ruter(områder) og tabeller. Mange av komponentene er i stand til å sortere objekter, trykke og utføre dra og slipp. Dette er bare noen av mulighetene som Swing innehar.

Utseende til Swing applikasjoner er lett å endre. Vi kan enkelt velge en annen utseende alt etter behov og ønske. Som for eksempel kan samme program både bruke Java eller Windows sitt utseende. En eksempel på applikasjon som bruker både Windows og Motif sitt utseende, se figur 3.2. Koden til denne applikasjonen, se side 32.

Swing applikasjoner kan vi lett implementere ved å importere pakken javax.swing. Figur 3.3 viser en enkelt swing applikasjon. Koden til denne applikasjonen, se side 33.

```
18 public class Utseende extends JFrame{
19
20   public Utseende()
21   {
22     this.setTitle("WindowsLookAndFeel");
23     JMenuBar menubar = new JMenuBar();
24
25     JMenu fileMenu = new JMenu("File");
26     fileMenu.setMnemonic('F');
27
28     JMenu editMenu = new JMenu("Edit");
29
30     JMenuItem openItem = new JMenuItem("Open");
31     openItem.setMnemonic('O');
32          openItem.setAccelerator(KeyStroke.getKeyStroke("control␣O"));
33
34     JMenuItem quitItem = new JMenuItem("Quit");
35     quitItem.setMnemonic('Q');
36          quitItem.setAccelerator(KeyStroke.getKeyStroke("control␣Q"));
37
38     fileMenu.add(openItem);
39     fileMenu.addSeparator();
40     fileMenu.add(quitItem);
41     menubar.add(fileMenu);
42     menubar.add(editMenu);
43
44     this.setLayout(new FlowLayout());
45     this.setJMenuBar(menubar);
46
47     this.getContentPane().setBackground(Color.WHITE);
48     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
49     this.setSize(450,300);
50     this.setVisible(true);
51   }
52
53   public static void main(String args[])
54   {
55     try {
56           UIManager.setLookAndFeel(
57             "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
58       } catch (Exception e) { }
59
60       Utseende utsee = new Utseende();
61   }
62 }
```

```
4  public class HalloVerdenSwing {
5
6    private static void createAndShowGUI() {
7          //Lager og setter opp vinduet
8          JFrame frame = new JFrame("HalloVerdenSwing");
9          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10
11         //Lager Hello World text
12         JLabel label = new JLabel("Hallo Verden");
13         frame.getContentPane().add(label);
14
15         //viser fram vinduet
16         frame.pack();
17         frame.setVisible(true);
18     }
19   public static void main(String[] args) {
20     javax.swing.SwingUtilities.invokeLater(new Runnable() {
21             public void run() {
22                 createAndShowGUI();
23             }
24         });
25
26   }
27
28 }
```



Figur 3.2: Motif og Windows-LookAndFeel

Figur 3.3: Hallo verden

3.1.3 AWT vs. Swing

AWT bruker inneværende tungvektige komponenter fra OS. Mens Swing tegner alle sine lettvektige komponenter selv. Hendelsene som kommer fra lettvektige komponenter blir analysert av Swing for å avgjøre hvilken komponent som genererte hendelsen. Siden Swing må tegne og analysere selv vil AWT være litt raskere enn Swing. Sannsynligvis vil ikke brukeren klare å merke noen forskjell i ytelse.

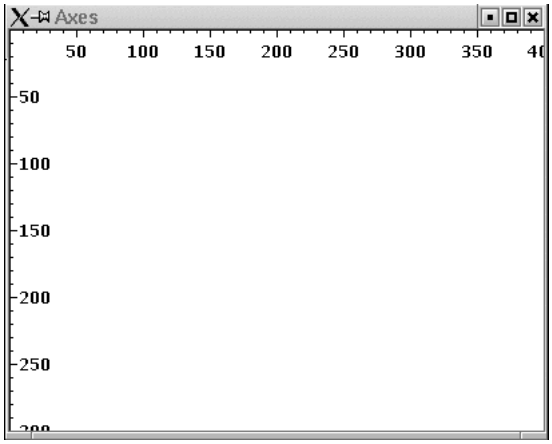
Den største fordelen med Swing er at den gir oss en fin mulighet til å bestemme over utseende for komponentene våre. Vi kan konfigurere utseende til en allerede eksisterende komponent, eller dersom vi ønsker har vi også muligheten til å lage egne komponenter fra bunnen av.

3.1.4 Vurdering/Sammenligning

«Heavyweight» komponenter er beslektet til sitt eget inneværende skjermressurs mens «Lightweight» komponenter er den som låner skjermen til å tegne sin egen komponent. Det er noen få viktige forskjeller mellom tungvekts og lettvekts komponenter. Vi har tidligere sagt at alle AWT komponenter er tungvektkomponenter og nesten alle Swing komponenter er lettvektskomponenter. Disse forskjellene synes tydelig når vi setter Swing komponenter sammen med AWT komponenter. Lettvekts komponenter har gjennomsynelige piksels mens en tungvekts komponent er ugjennomsiktig. Lettvekts komponenter ser ut til å være ikke rektangulære i formen grunnet dens evne til å sette gjennomsynelige områder. En tungvekts komponent er begrenset til å være rektangulær[34].

3.2 Tegning i Swing/AWT

Tegninger i Java lages relativt lett ved at vi kan kalle på noen metoder fra standard biblioteket. For tegning i Java er det viktig å kjenne til en av de mest betydningsfulle objektinstanser som blir opprettet under grafikk oppgaver. Dette objektet er en instans av java.awt.Graphics klassen og refererer til et område på skjermen som en Applet eller en JFrame utgjør. En «graphics» objektinstans gir oss muligheten til å bruke alle de tegneoperasjonene som er tilgjengelig i den klassen. Den inneholder også «kontekstuell» informasjon



Figur 3.4: Venstre hånds koordinatsystem [33]

om tegnepanelets oppmerkede område, valgt farge, overføringsmodus samt tekstfont.

Ofte er grafikk instansen fått fra «Java window manager» som et resultat av en tegne etterspørsel. Denne instansen er også sendt videre til vårt komponents paint eller update-metode. Det finnes også «Graphics2D» som er en sub klasse av «Graphics» og den inneholder et meget rikere sett av tegneoperasjonene. For å bruke «Graphics2D» kan man enkelt kaste et «Graphics» objekt til «Graphics2D» objekt. Se koden vist under for tegning av en firekant for å vite hvordan man kaster. «Graphics2D» inkluderer pennbredder, sprunget linjer, bilde operasjoner, hellingsgrad av farge over mønstre, mulighet til å bruk vilkårlige lokal fonter, flyte tall koordinatsystem og mange koordinat transformasjon operasjoner.

Koordinat systemet starter fra (0,0) øverst på hjørnet til venstre av det tegneområdet med x og y som øker seg til de kommer ved Appleten sin bredde og høyde. Koordinatene er målt i piksels, med x som øker horisontal til høyre og y øker vertikalt ned. Se på figur3.4.

3.2.1 «Frihåndstegning»

I denne oppgaven må det lages mange logiske porter som er utformet av matematiske former. Derfor i dette avsnittet skal vi se på frihåndstegning med Java. Java gir oss mange muligheter til å utforme en enkelt form. Vi skal ta utgangspunkt i en enkelt form som er en firkant og se på de forskjellige mulighetene vi har.

Som nevnt finnes det flere måter for å lage en firkant i Java. For å lage en firekant kan vi bruke drawRect-metoden fra klassen Graphics, eller det finnes også forhånds definerte geometriske former i pakken java.awt.geom, eller kan man bruke «GeneralPath» og bygge stier som i postscript. Det finnes sikkert

flere andre metoder for å lage en firekant som ikke er nevnt her.

Med AWT tegnet vi vanligvis en form ved å kalle på drawXXX-metoden eller fillXXX-metoden fra «Graphics»-objektet. Men i Java2D kan vi lage et «Shape»-objekt og kalle på tegne metoden eller fylle metoden fra «Graphics2D»-objektet, ved å sende «Shape»-objektet som en parameter. Shape er et grensesnitt i Java som kan også benyttes til å definere egne former. Figur 3.5 viser en tegning av firekanter. De firkantene er tegnet ved å benytte noen av de metodene som finnes i Java. Koden for dette vises under figuren på side 37.

3.2.2 Text

java.awt.font klasse blir brukt til å opprette Font-objekter som bestemmer fonten for å skrive tekst, merker lapper(labels), tekstfelt, knapper osv.

Det er definert 5 logiske fonter i Java2D som følger under:

- Dialog
- DialogInput
- Monospaced
- Serif
- SansSerif

Disse fontene er tilgjengelig på alle Java plattformer og har en egenskap som navnet beskriver for noen av de fonttypene som ligger under. En Serif font er en liknende font type som Times New Roman og vanligvis er brukt til å skrive ut på Standard output med print metoden. Sans Serif font er mer typisk brukt til GUI. Fonter har også en viktig egenskap som størrelse og stil. Stiler er fet eller kursiv.

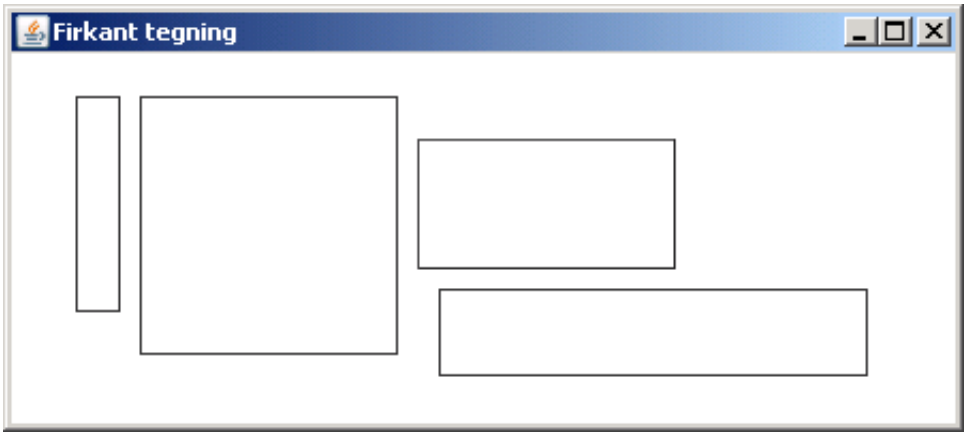
Standard font for Java 2D er 12pt Dialog. Denne fonten har en typisk punktstørrelse for lesende tekst på en normal 72-120 DPI display device. En applikasjon kan direkte lage en instance av denne font ved å spesifisere følgende linje.

```
Font font = new Font("Dialog", Font.PLAIN, 12);
```

I tillegg til de logiske fontene, Java gir også tilgang til andre fonter som er installert på maskinen. Navnet til alle de tilgjengelige fonter kan vi finne ut ved å gjøre følgende[15]:

```
GraphicsEnvironment ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
String []fontFamilies = ge.getAvailableFontFamilyNames();
```

Et eksempel på bruk av fonter og farger, se side 38.



Figur 3.5: Tegning av firkanter

```
10 import javax.swing.JPanel;
11
12 public class Draw extends JPanel {
13
14   Rectangle2D.Double rec = new Rectangle2D.Double(60, 20, 120, 120);
15   GeneralPath path = new GeneralPath();
16
17
18   public void paintComponent(Graphics g) {
19
20     g.drawRect(30, 20, 20, 100);
21     Graphics2D g2 = (Graphics2D) g;
22     g2.draw(rec);
23     g2.draw3DRect(190, 40, 120, 60, true);
24
25     path.moveTo(400, 110);
26     path.lineTo(400, 150);
27     path.lineTo(200, 150);
28     path.lineTo(200, 110);
29     path.closePath();
30     g2.draw(path);
31
32
33   }
34
35   public static void main(String args[]) {
36     JFrame frame = new JFrame("Firkant tegning");
37     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
38     Draw draw = new Draw();
39     // Add the ubiquitous "Hello World" label.
40     frame.getContentPane().add(draw);
41
42     // Display the window.
43     frame.pack();
44     frame.setSize(450,200);
45     frame.setVisible(true);
46     frame.setBackground(Color.WHITE);
```

```
18 public class TextFont extends JFrame{
19
20   public TextFont()
21   {
22     super("Text/Font");
23     JPanel pane = new JPanel();
24     pane.setBackground(Color.WHITE);
25
26     JLabel title = new JLabel("Simulering av");
27     title.setFont(new Font("Eras Bold ITC", Font.BOLD, 48));
28
29     JLabel title2 = new JLabel("Digitale Kretser");
30     title2.setFont(new Font("Serif", Font.BOLD, 48));
31     title2.setForeground(Color.GREEN);
32
33     JLabel title3 = new JLabel("i Java");
34     title3.setFont(new Font("Forte", Font.BOLD, 48));
35     title3.setForeground(Color.BLUE);
36
37     pane.add(title);
38     pane.add(title2);
39     pane.add(title3);
40
41     this.add(pane);
42     this.getContentPane().setBackground(Color.WHITE);
43     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
44     this.setSize(450,250);
45     this.setVisible(true);
46   }
47
48   public static void main(String[] args) {
49     // TODO Auto-generated method stub
50     TextFont textfont = new TextFont();
51   }
52
53 }
```

Figur 3.6: Demonstrasjon av fonter med farger

3.2.3 Zooming

Java gir en del muligheter for å zoom in og ut et bilde eller en tegning som er tegnet av geometriske former. Det er heller ikke noe vanskelig å få til denne funksjonaliteten i Java. Denne funksjonaliteten trenger vi for å zoome inn eller ut enkelte porter og kretstegninger i oppgaven. Viktigste av denne funksjonaliteten er at vi må kunne zoome inn eller ut uten å tape kvaliteten av portene. Siden det er flere måter å gjøre dette på, skal vi se på hvilken som er aktuelt og gir oss mulighet til å zoome inn eller ut uten å tape kvaliteten.

Figur 3.7 viser et eksempel på zoome inn et rektangel. Koden for dette vises under figuren.

Et eksempel på forstørring av et bilde, se figur 3.9. Dette bildet er forstørret ved hjelp av Graphics2D.draw-metoden. Koden for dette, se side 41.

Det finnes mange andre muligheter for å skalere et bilde eller geometriske former og her skal vi se på noen av dem: Metode 1:

```
//scale up using coordinates
g.drawImage(img,
0, 0, w*3, h*3, /* dst rectangle */
0, 0, w, h, /* src area of image */
null);
```

Metode 2:

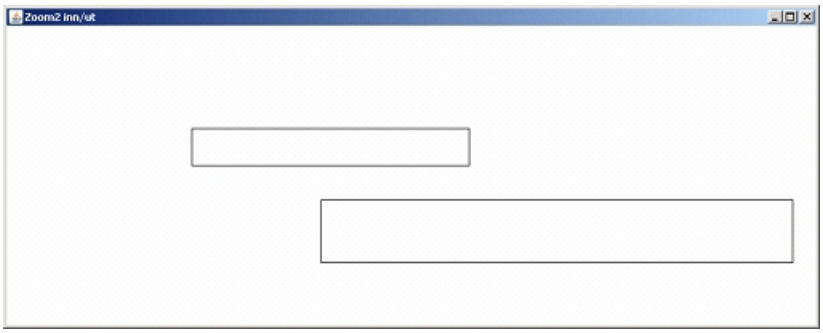
```
/* scale down using transform */
g2.drawImage(img, AffineTransform.getScaleInstance(1.7, 1.7), null);
```

Metode 3:

```
/* scale up using transform Op and BICUBIC interpolation */
AffineTransform at = AffineTransform.getScaleInstance(1.5,1.5);
AffineTransformOp aop = new AffineTransformOp(at,
                        AffineTransformOp.TYPE_BICUBIC);
g2.drawImage(img, aop, 0, 0);
```

3.2.4 Rammer

Java har mange forskjellige type rammer som Applet, Canvas, Component, Container, Dialog, Frame, JApplet, JComponent, JContainer, JDialog, JInternalFrame, JPanel, JFrame, JWindow, Panel og Window. Hovedforskjellene mellom dem er at noen er AWT komponent og det eksisterer også en forbedret versjon i Swing, noen av de komponentene er ikke synlig og har også forskjellige måter å sette layout til. Vi skal bare se på de mest aktuelle rammer for denne oppgaven.



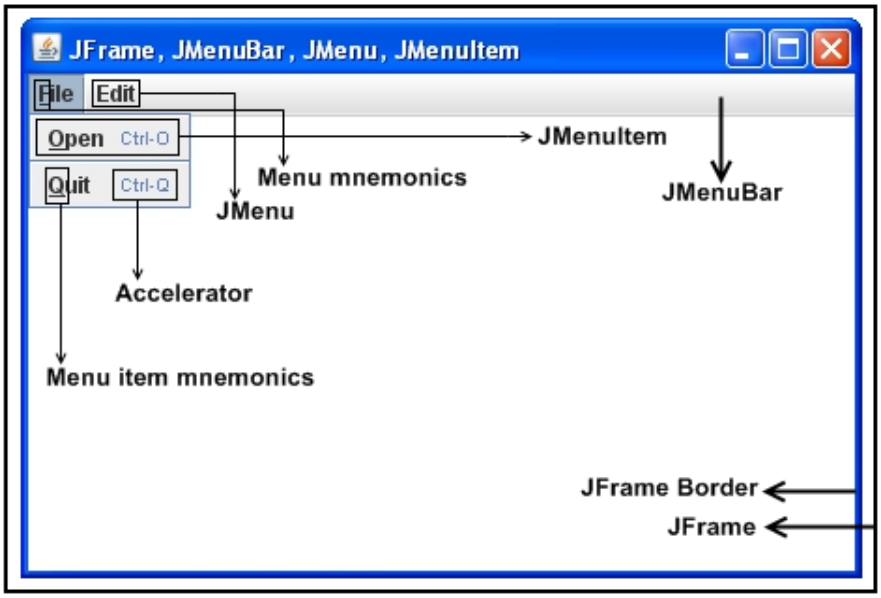
Figur 3.7: Zooming av Geometriskeformer.

```
26 public class RectangularZoom extends JFrame {
27   BufferedImage img;
28   int w, h;
29
30   public RectangularZoom() {
31     super("RectangularZoom");
32
33     JPanel panel = new JPanel(null) {
34       public void paintComponent(Graphics g) {
35         Graphics2D g2 = (Graphics2D) g;
36         g2.setPaint(Color.WHITE);
37         g2.fillRect(0, 0, getWidth(), getHeight());
38         GeneralPath path = new GeneralPath();
39
40         g2.setPaint(Color.BLACK);
41         path.moveTo(500, 110);
42         path.lineTo(500, 150);
43         path.lineTo(200, 150);
44         path.lineTo(200, 110);
45         path.closePath();
46         g2.draw(path);
47         Shape sh = path.createTransformedShape
48         (AffineTransform.getScaleInstance(1.7, 1.7));
49         g2.draw(sh);
50       }
51     };
52     panel.setPreferredSize(new Dimension(510, 350));
53     this.add(panel);
54     this.getContentPane().setBackground(Color.WHITE);
55     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
56     this.setSize(350, 350);
57     this.setVisible(true);
58   }
59   public static void main(String[] args) {
60     // TODO Auto-generated method stub
61     RectangularZoom zoom2 = new RectangularZoom();
62   }
63
64 }
```

```
26 public class AppleZoom extends JFrame {
27   BufferedImage img;
28   int w, h;
29
30   public AppleZoom() {
31     super("AppleZoom");
32     try {
33       img = ImageIO
34           .read(new File(
35     "G:\\Master Oppgave\\WorkSpace\\TestProject\\green_apple_logo.jpg"));
36     } catch (IOException e) {
37       // TODO Auto-generated catch block
38       System.out.println("IMAGE ERROR " + e.getMessage());
39     }
40
41     w = img.getWidth(null);
42     h = img.getHeight(null);
43
44     JPanel panel = new JPanel(null) {
45
46       public void paintComponent(Graphics g) {
47         Graphics2D g2 = (Graphics2D) g;
48         g2.setPaint(Color.WHITE);
49         g2.fillRect(0, 0, getWidth(), getHeight());
50
51         // scale up using coordinates
52          g.drawImage(img,
53          0, 0, w*3, h*3, /* dst rectangle */
54          0, 0, w, h, /* src area of image */
55          null);
56       }
57     };
58
59     panel.setPreferredSize(new Dimension(510, 350));
60
61     this.add(panel);
62     this.getContentPane().setBackground(Color.WHITE);
63     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
64     this.setSize(350, 350);
65     this.setVisible(true);
66   }
67
68   /**
69    * @param args
70    */
71   public static void main(String[] args) {
72     // TODO Auto-generated method stub
73     AppleZoom zoom2 = new AppleZoom();
74   }
75
76 }
```

Figur 3.8: Eple uten Zoom.

Figur 3.9: Eple Med Zoom.



Figur 3.10: JFrame med Menyer

3.2.4.1 JFrame

En «JFrame»-ramme er et hovedvindu med en tittel og en kant. Størrelsen til den rammen inkluderer inn området som er satt av med kanten. Dimensjonene til kanten kan fås tak i ved å kalle på metoden getInsets. Siden kant området er inkludert i totalstørrelsen, skjuler den en porsjon av rammen.

For å legge til komponenter til en «JFrame» ramme må vi spesifisere hvilken type layout vi skal bruke. Layout vil si hvilket oppsett, for eksempel måten komponentene skal bli plassert på. Det finnes mange forskjellige type layout i Java som BorderLayout, BoxLayout, CardLayout, FlowLayout, GridBagLayout, GridLayout, GroupLayout, SpringLayout. Vi kan også definere vår egen layout eller kan man også la være å bruke layout. Dersom man ikke bruker noen layout må vi gjøre en fullstendig plassering for komponentene, det vil si vi må oppgi x, y, bredde og høyde. En Layoutmanager blir brukt til å spesifisere hvilken type layout man ønsker å bruke. Figur 3.10 viser et JFrame med menyer. Koden til denne applikasjonen, se side 44.

3.2.4.2 LayoutManager

Et oppsett håndterer i Java er et objekt som implementerer LayoutManager eller LayoutManager2-grensesnittet og fastsetter størrelsen og posisjonen av komponentene innenfor en «Container»-beholder. Selv om komponenter kan si størrelsen og plassering hint, det er en oppsett håndterer som kan si det

```
16 public class FrameAndMenu extends JFrame{
17
18   public FrameAndMenu()
19   {
20     this.setTitle("JFrame, JMenuBar, JMenu, JMenuItem");
21     JLabel label = new JLabel("Test");
22
23     JMenuBar menubar = new JMenuBar();
24
25     JMenu fileMenu = new JMenu("File");
26     fileMenu.setMnemonic('F');
27
28     JMenu editMenu = new JMenu("Edit");
29
30     JMenuItem openItem = new JMenuItem("Open");
31     openItem.setMnemonic('O');
32         openItem.setAccelerator(KeyStroke.getKeyStroke("control O"));
33
34     JMenuItem quitItem = new JMenuItem("Quit");
35     quitItem.setMnemonic('Q');
36         quitItem.setAccelerator(KeyStroke.getKeyStroke("control Q"));
37
38     fileMenu.add(openItem);
39     fileMenu.addSeparator();
40     fileMenu.add(quitItem);
41     menubar.add(fileMenu);
42     menubar.add(editMenu);
43
44     this.setLayout(new FlowLayout());
45     this.setJMenuBar(menubar);
46     this.add(label);
47
48     this.getContentPane().setBackground(Color.WHITE);
49     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
50     this.setSize(450,300);
51     this.setVisible(true);
52   }
53
54   public static void main(String args[])
55   {
56     FrameAndMenu ramme = new FrameAndMenu();
57   }
58 }
```

siste ordet når det gjelder dette. LayoutManager2 er en minimal utvidelse av LayoutManager.

Det finnes forhånds definerte oppsett håndterer i Java under pakken java.awt og javax.swing:

- ScrollPaneLayout
- BorderLayout
- BoxLayout
- CardLayout
- FlowLayout
- GridBagLayout
- GridLayout
- GroupLayout
- SpringLayout

Det finnes også tredje part sin oppsett håndterer og noen av de meste populære er[46]:

- TableLayout
- MiGLayout
- Karsten Lentzsch's FormLayout

Dersom vi ikke er innterresert i lære oss alle de detaljene av et oppsett håndterer, kan vi bruke en «GroupLayout» -oppsett håndterer med et verktøy for oppbygging av GUI. Et slikt verktøy er NetBeans IDE. Ved bruk av et slikt verktøy kan komme unødvendige koder som vil gjøre vanskelig for oss å debugge. Dersom vi ikke ønsker en slik løsning anbefaler Sun til å bruke GirdBagLayout som er den mest fleksible og sterkeste oppsett håndterer[46].

3.2.4.3 Rullefelt

Brukeren kan dra en knapp eller trykke på en pil, vertikalt eller horisontalt eller begge for å se området på skjermen som ikke er synlig, dette kalles «Scrollbars». I AWT får vi scrollbars ved å legge komponenten til en Scrollpane og ScrollPane til en «Container». Alle typer Windows, Panels, Frames and Dialog bokser i Java er «Containers». Normalt kalles alle grafiske komponenter som har sub komponenter for «Containers», men klasser holder og organiserer andre objekter som «trees», «bags» og «Vectors» kalles for «collections» for å unngå forvirring. I Swing bruker vi JScrollPane.

JScrollPane fungerer med ScrollPaneLayout og gir mulighet til å vise rullegardinen bare ved når det trenges.

Dersom vi legger et Panel med ingen layout til JScrollPane, vil det ikke komme opp rullegardinen automatisk når vi trenger det. I Java bruker vi ordet null layout dersom vi ikke spesifiserer noen layout. Rullegardinen vil bare komme opp når en foretrukket størrelse til panelet er større enn JScrollPane. Siden null-Layout i Java ikke er et oppsett håndterer som er ansvarlig for å kalkulerer eller innsette foretrukket størrelsen til panelet, må vi kalle selv metoden «setPreferredSize» ved å gi nye størrelsen. setBounds-metoden kan brukes for å plassere et komponent til det panelet og til slutt må revalidate-metoden kalles for å oppdatere skjermbildet. For å unngå å kalkulere selv størrelsen er det anbefalt å bruke et oppsett håndterer som gjør dette selv.

Et eksempel med uten «setPreferredSize», se side 47.

Siden oppgaven vår går ut på å tegne kretser med mus er jeg nødt til å bruke et null-Layout til det meste i prosjektet. Fordi punktene for plassering av elementer og tegning av strøm kretser blir sendt som x, y punkter i piksels fra musa. Dermed har jeg laget en eksempel med et null-Layout, se side 48.

Forskjellen mellom Rullegardin2 og Rullegardin er at Rullegardin2 klassen bruker GridBagLayout og dermed kan vi legge til noe på panelet uten å bekymre seg om størrelsen. Fordi dersom det går utover det som er synlig på skjermen vil det automatisk dukke opp rullegardin. Men i klassen Rullegardin bruker jeg ikke noen layout håndterer og dermed må vi kalkulere selv om plassen på panelet er nok for det vi legger til. Dersom det ikke er nok må vi kalle på metoden setPreferredSize med nye størrelsen og da vil det dukke opp rullegardin. Det er viktig å huske på å kalle metoden revalidate etter at vi legger til noe nytt på GUI, fordi da oppdateres skjermbildet og det nye skjermbildet vises med endringer.

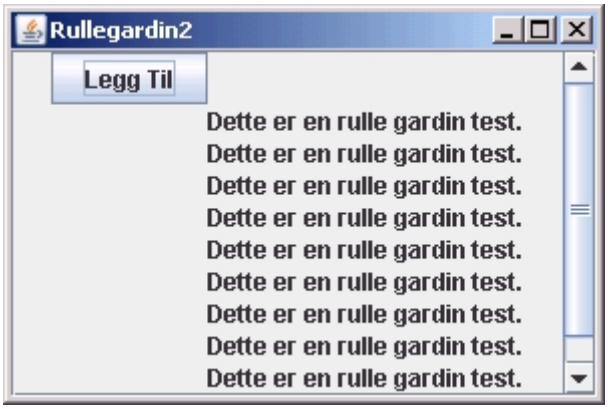
3.2.4.4 Menyer

En meny lar brukeren velge en av de adskillige alternativer på en plass sparing måte. Andre komponenter som lar brukeren velge et av flere valg er «combo» boks, lister, radio knapper, «spinners» og verktøylinje.

Det finnes to hovedtyper meny, «dropdown» og «popup». «Dropdown» er den tradisjonelle menyen som er plassert over toppen av et vindu i en meny bar og viser under meny navnet. «Popup» menyen kommer fram når brukeren klikker, for eksempel ved et høyre mus klikk eller over en komponent som kan håndtere en popup etterspørsel. Java gir oss mulighet til å implementere alle disse typer meny.

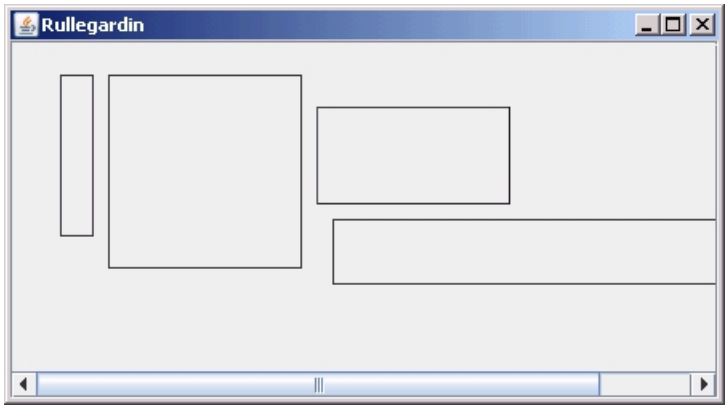
En meny bar kan bli lagt til toppen av en topp nivå «container»(beholder), for eksempel JFrame, JApplet eller JDialog.

«Dropdown» menyer har tre deler:



Figur 3.11: Rullegardin2 uten «setPreferredSize».

```
25 public class Rullegardin2 extends JFrame implements ActionListener{
26   GridBagConstraints c;
27   JPanel panel;
28   int count = 1;
29   public Rullegardin2()
30   {
31     super("Rullegardin2");
32     c = new GridBagConstraints();
33     panel = new JPanel(new GridBagLayout());
34     Container contant = this.getContentPane();
35
36     JButton button = new JButton("Legg Til");
37     button.addActionListener(this);
38
39     c.fill = GridBagConstraints.HORIZONTAL;
40     c.anchor = GridBagConstraints.PAGE_START;
41     panel.add(button, c);
42
43     JScrollPane jscroll = new JScrollPane(panel);
44     contant.add(jscroll);
45     this.getContentPane().setBackground(Color.WHITE);
46     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
47     this.setSize(300,200);
48     this.setVisible(true);
49   }
50   public void actionPerformed(ActionEvent e) {
51     JLabel label = new JLabel("Dette er en rulle gardin test.");
52     c.fill = GridBagConstraints.HORIZONTAL;
53     c.gridx = 10;
54     c.gridy = count;
55     panel.add(label, c);
56     panel.revalidate();
57     count++;
58   }
59     public static void main(String[] args) {
60       Rullegardin2 rulle2 = new Rullegardin2();
61   }
62 }
```



Figur 3.12: Rullegardin med «setPreferredSize».

```
16 public class Rullegardin extends JFrame{
17   public Rullegardin()
18   {
19     super("Rullegardin");
20     JPanel panel = new JPanel(null){
21       public void paintComponent(Graphics g) {
22         Rectangle2D.Double rec = new Rectangle2D.Double(60, 20, 120, 120);
23         GeneralPath path = new GeneralPath();
24         g.drawRect(30, 20, 20, 100);
25         Graphics2D g2 = (Graphics2D) g;
26         g2.draw(rec);
27         g2.draw3DRect(190, 40, 120, 60, true);
28
29         path.moveTo(500, 110);
30         path.lineTo(500, 150);
31         path.lineTo(200, 150);
32         path.lineTo(200, 110);
33         path.closePath();
34         g2.draw(path);
35         this.repaint();
36       }
37     };
38     panel.setPreferredSize(new Dimension(510,200));
39     JScrollPane jscroll = new JScrollPane(panel);
40
41     this.add(jscroll);
42     this.getContentPane().setBackground(Color.WHITE);
43     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
44     this.setSize(450,250);
45     this.setVisible(true);
46   }
47
48   public static void main(String[] args) {
49     Rullegardin rulle = new Rullegardin();
50   }
51 }
```

1. JMenuBar Plasser på toppen av «container». Menyen blir lagt til meny bar

2. JMenu har et navn og inneholder menyvalgene som blir vist i en vertikal liste. Den blir lagt til JMenuBar.

3. JMenuItems og separatorer er lagt til hver meny. Funksjoner er vanligvis tekst «knapper», men også kan ha ikoner, kontrollfelter, radioknapper, eller kan være hierarkisk sub menyer. JMenuItems blir lagt til JMenu.

I Java har man også mulighet til å påkalle menyer eller funksjoner fra tastaturet ved å tilknytte karakter. Dette kan deles inni tre deler:

1. «Menu mnemonics» kan bli brukt til å åpne en meny ved å taste en enkel karakter som er tilknyttet med menyen sammen med et operativsystem definert nøkkel. For eksempel i Windows bruker vi «Alt + f» for å åpne en fil meny.`fileMenu.setMnemonic('F');`

2. «Menu item mnemonics» blir brukt til å velge ut en bestemt funksjon når dets meny er allerede åpent. Typisk svarer karakteren til den første bokstav eller et signifikant til navnet og det tegnet blir understrekt.

3. «Accelerator» - Akselerator nøkkel kombinasjoner er direkte brukt til å påkalle et meny valg uten å åpne menyen. For eksempel Ctrl + C utfører kopiering. Akselerator nøkkel alternativer er vist til høyre for menyvalg navnet.

Bilden 3.10 viser dette.

3.3 Brukerinteraksjon

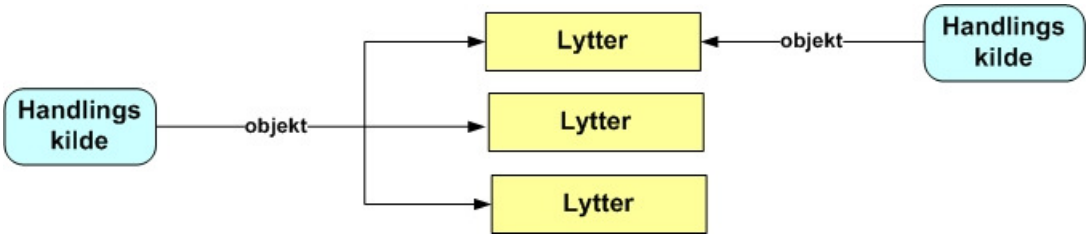
Dere har sikkert noe erfaring med lyttere i Java så langt. Vi kan se på en veldig enkel eksempel med en knapp som lager lyd når vi trykker den, se side 50.

Beep klassen implementerer «ActionListner»-grensesnittet som inneholder en metode og den heter «actionPerformed». Siden klassen Beep implementerer «actionPerformed» kan en Beep objekt lytte på handlinger på knappen. Når vi har lagt til «addActionListener» til Beep knappen, vil den lytte på enhver handling som vi utfører på knappen.

Lyttere i Java er veldig kraftig og fleksible. Flere lyttere kan lytte på alle slags handlinger fra flere handlings kilder. Det vil si vi kan ha mange til mange relasjon. For eksempel et program kan lage en lytter for hvert handlings kilde. Eller et program kan ha en enkel lytter for alle handlingene som kommer fra forskjellige kilder. Et program kan også ha mer enn en lytter

Figur 3.13: Lyttertest.

```
14 public class Beep extends JFrame implements ActionListener{
15   public Beep() {
16
17     this.setTitle("Lytter test");
18     JButton beep = new JButton("Beep");
19     beep.addActionListener(this);
20
21     this.setLayout(new FlowLayout());
22     this.add(beep);
23     this.setBackground(Color.WHITE);
24     this.setSize(200, 80);
25     this.setVisible(true);
26     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
27   }
28
29   public static void main(String args[])
30   {
31     Beep beepframe = new Beep();
32   }
33
34   @Override
35   public void actionPerformed(ActionEvent arg0) {
36     // TODO Auto-generated method stub
37     Toolkit.getDefaultToolkit().beep();
38
39   }
40 }
```



Figur 3.14: Lytterdiagram

for en enkel type handling som kommer fra en enkelt handslings kilde, se figur 3.14.

Hver handling blir representert av et objekt som gir informasjon om handlingen og handlingskilden. For eksempel objektet kan være et «ActionEvent» objekt og handlingskilden kan være en «JButton».

Når en handling blir utført vil den gå til den sin handlings håndterer. For eksempel hvis det en type handling av «ActionEvent» vil den gå til «actionPerformed» metoden som «ActionEvent» objekt og den vil da inneholde også informasjon om handlingskilden.

3.3.1 Lyttere

Den viktigste regelen man må huske om lyttere er at de må kunne utføre veldig raskt. Fordi all tegning og lytting metoder kjøres i samme tråd. En treg metode kan gjøre programmet til å se ut som ikke svarer og langsom tegning ved kall på «repaint». Dersom vi må gjennomføre noe langtrukken drift av en handling, er det viktig å starte enda en egen tråd for denne operasjonen.

Det finnes mange måter å implementere lyttere på. Sun rekommanderer ikke noen spesielle måte på grunn av at en løsning ikke egner seg godt til alle situasjoner, men de gir bare noen gode råd til utviklere.

Vi kunne velge å implementere separate klasser for forskjellige typer av lyttere. Det kan være en lett arkitektur til å vedlikeholde, men mange klasser kan også bety redusert ytelse. Når vi designer et program kunne vi unngå å ha lyttere i en «public» klasse, men ett eller annet sted mer gjemt. Det vil si en «private» implementering er mer sikker implementering. Hvis vi har en meget spesifikk type av en enkel lytter, kunne vi unngå å lage en klasse i det hele tatt ved å bruke «Eventhandler» klasse.

3.3.1.1 Event Object

Hver handlings lytter metoden tar i mot et argument. Det argumentet varierer med hvilken type handlings lytter det er. For eksempel hvis det er en mus lytter vil det argumentet være «MouseEvent» objekt, hvor «MouseEvent» objekt er en indirekte sub klasse av «EventObject».

Den «EventObject» klassen definerer en veldig nyttig metode som heter getSource.

- Object getSource-metoden returnerer det objektet som har sendt fra handlingen.

Metoden getSoure returnerer et objekt. Det finnes også en annen metode i «ComponentEvent» klasse som heter getComponent. Det er nesten likt som getSource metode og returnerer det objektet som sendte handlingen. Forskjellen er at getComponent metoden returnerer alltid en komponent.

Ofte en «event» klasse definerer metoder som returnerer informasjon om den handlingen. For eksempel en «MouseEvent» objekt inneholder informasjon om hvor det oppstod handling, hvor mange klikk brukeren utførte og hvilket tegn ble trykket osv.

3.3.1.2 Lavt nivå og semantisk hendelse

En handlig kan deles i to grupper, lavt nivå og semantiskhendelse. På engelsk heter det «Low-Level Events and Semantic Events». Lavt nivå handling oppstår ved os-systemet eller lav nivå input og resten av alt er en semantisk. Eksempler på lavt nivå handlinger er mus og tastatur. Semantisk handlinger er «action» og «item events». For eksempel det kunne være noe som brukeren utløser seg selv, ved å trykke på en knapp osv.

3.3.1.3 Event Adapter

Noen lytter grensesnitt har mer enn en metode. For eksempel, mus lytter grensesnitt som er «MouseListener» inneholder fem metoder: mousePressed, mouseReleased, mouseEntered, mouseExited og mouseClicked. Dersom vi har implementert direkte «MouseListener» i vårt klasse og selv om vi trenger å ta hensyn til bare mus klikket må vi implementere alle fem metoder. De metodene som vi ikke trenger å ta hensyn til kan være tomme.

Et eksempel på en klasse som implementerer lytter grensesnittet direkte.

```
4  public class Mouse implements MouseListener {
5
6    public void mouseClicked(MouseEvent e) {
7      System.out.println("mouseClicked");
8    }
9
10   public void mouseEntered(MouseEvent e) {
11   }
12
13   public void mouseExited(MouseEvent e) {
14   }
15
16   public void mousePressed(MouseEvent e) {
17   }
18
19   public void mouseReleased(MouseEvent e) {
20   }
21
22 }
```

Resultat av slik koding med tomme metoder vil være tungt å lese og vedlikeholde. For å unngå slike tomme metoder gir API'et mulighet til å inkludere en adapter klasse for hver lytter grensesnitt med flere enn en metode. For eksempel «MouseAdapter»-klasse implementerer «MouseListener»-grensesnittet. En adapter klasse implementerer en tom versjon for alle grense-

snitt metoder. For å bruker en adapter må man lage en sub klasse av den og kjøre over med bare de metodene som vi trenger.

En eksempel med en forbedret versjon av klasse Mouse:

```
5  public class Mouse2 extends MouseAdapter {
6
7        public void mouseClicked(MouseEvent e) {
8          System.out.println("mouseClicked");
9        }
10   }
```

3.4 Konklusjon

Swing er en utvidet GUI-verktøy basert på AWT. Både AWT og Swing inneholder de nyttige funksjoner som vil dekke behovene til denne oppgaven. Et eksempel på dette er grafiske komponenter med lyttefunksjon.

Del II

Prosjektet

Kapittel 4

JBuilder kontra Eclipse

Dette programmet skulle utvikles i Java og da stod valget for vår[1] del mellom to utviklingsverktøy: JBuilder og Eclipse.

Vår problemstilling her var at vi skulle velge det beste utviklingsverktøyet for oppgaven. Fra før var vi mest kjent med JBuilder ved Høgskolen i Oslo. Andre skoler har begynt å innføre Eclipse som standard verktøy for utvikling.

4.1 JBuilder

JBuilder ble utviklet av Borland sent i 1995. De første versjonene var stort sett utviklet i Delphi. Det var ikke før i versjon 3.5 at den var skrevet i Java. I 1997 kunne brukerne legge til «add-ons» som for eksempel OpenTools og tilpasse verktøyet JBuilder til eget formål. Med denne muligheten lagde Oracle på den tiden sitt eget utviklingsverktøy kalt JDeveloper. Populariteten økte og JBuilder fikk flere plugins. Det kommer stadig nye versjoner av dette produktet med stadig høyere kvalitet og nye funksjonaliteter. Den nyeste versjon er JBuilder 2007[28].

Noen fordeler med JBuilder er:

- Kan jobbe med flere applikasjons-servere.
- Kan brukes for å utvikle web-servere og JSPer.
- Har grafiske verktøy som aksellerer Web-Java — og Open Source utvikling.
- Optimizeit som leverer avansert debugging og ytelsesjustering.
- LiveSource som er et UML-modelleringsverktøy som sikrer at koden og modulene alltid er synkroniserte.

[1] Dette kapittelet er utarbeidet i samarbeid med min medstudent. Derfor benyttes «vi» formen.

- TeamInsight som hjelper å håndtere komplekse prosjekter på flere lokasjoner.

4.2 Eclipse

Eclipse er et open source programmerings-rammeverk skrevet i Java. Det begynte som et IBM Candada-prosjekt. Programmet var skrevet av OTI (Object Technology International) og skulle erstatte VisualAge, et program de hadde skrevet selv[27].

I november 2001 ble det dannet et konsortium som skulle føre videre Eclipse som open source, og i 2003 ble Eclipse Foundation dannet[25]. Opprinnelig var Eclipse lansert under CPL (Common Public License), en lisens utviklet av IBM, men ble relansert under EPL (Eclipse Public License) utviklet av Eclipse Foundation.

CPL inneholder en patent-klausul for å forhindre folk i å slippe ut modifikasjoner av programmer å ta betalt for det[23]. I EPL er denne patent-klausulen fjernet og blitt mer business-vennlig. Mottakeren av EPL-lisensen kan modifisere og distribuere sine egne versjoner av progammet. Men EPL har svakere copyleft enn GNU GPL (GNU General Public License)[24]. GNU GPL er den mest kjente free-software-lisens og har en sterk copyleft, som er en form for lisens hvor innehaveren av et program tillater andre å bruke, modifisere og redistribuere kode uten å ta betalt for det, forutsatt at innehaverens navn er tatt med i programmet [26].

4.3 Konklusjon

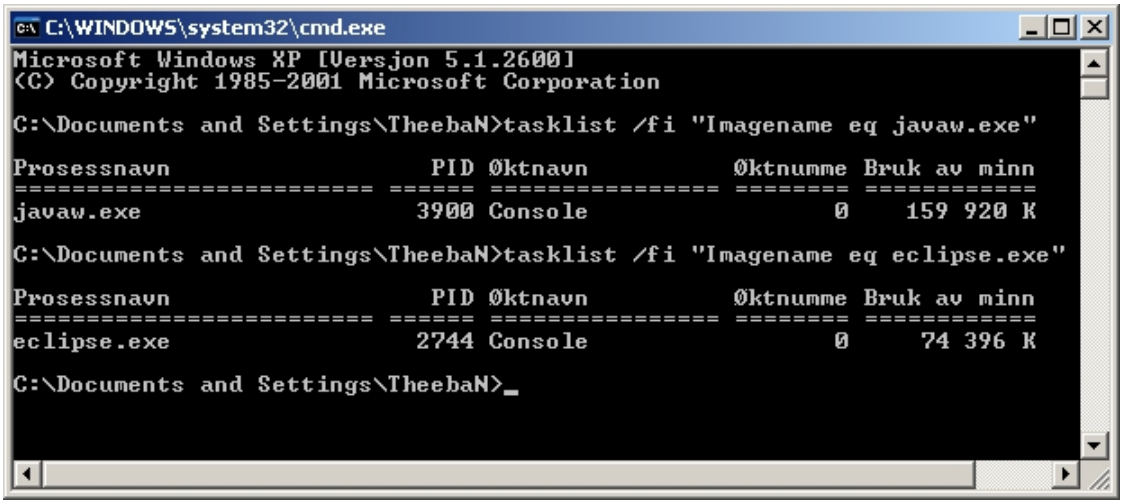
Eclipse har støtte for mange plug-ins siden det er et open source verktøy. Plug-insene varierer fra aspekt-orientert utvikling til en innebygd MP3-spiller.

Både Eclipse og JBuilder støtter viktig funksjonalitet som syntakssjekking, assistanse for kode, kode-generering, samt støtte for refaktoring, Ant enhetstesting og CVS[30].

Siden Eclipse har mye støtte fra open source-samfunnet blir det stadig utviklet mange nye plugins. JBuilder støtter også plugins men har ikke åpen kildekode, og har færre plugins enn Eclipse. JBuilder 2007 som er den nyeste versjonen er nå bygget på Eclipse.

Eclipse er et utviklingsverktøy for flere programmeringsspråk og LaTeXsom vi bruker for å skrive masteroppgaven. Figur 4.1 viser en sammenligning av hvor mye ressurser hvert av de utviklingsverktøyene bruker.

Sammenligningen viser at JBuilder krever betydelig mer ressurser. Vi konkluderer dette avsnittet med å velge Eclipse for denne masteroppgaven.



Figur 4.1: Minneforbruket til Eclipse og JBuilder

Kapittel 5

Tegning av porter

Dette kapitlet handler om utseendet av portene og hvordan Java skal brukes til å tegne disse. Vi skal også se litt på hva Java tilbyr for å konstruere slike tegninger. Under eksemplene brukes kun de tre mest ulike portene AND, XNOR og NOT som utgangspunkt. Siden det er kun de som er mest forskjellige fra de andre og mest vanskelige.

5.1 Utseende

Det var viktig å finne en konvensjon til utseendet av porter. Denne finnes og heter IEC 60617, men standarddokumentasjonen er lisensiert og koster penger. Derfor valgte vi[1] å undersøke hos andre kilder som Wikipedia og Digital Works for å finne det riktige utseendet for portene i Digital Circuit.

Nettsiden Wikipedia hadde gode tegninger under «Circuit elements». På figur 5.1 vises det en AND-port fra Wikipedia. Det ble også oppdaget at det finnes en annen utseende til portene enn de vi er vant med[22]. Denne spesielle utseende til AND-porten kan sees på figur 5.2. Den vi er vant med heter «Military»-and og det er denne formen vi valgte å bruke.

På figur 5.3 vises utseendet til noen porter hos Digital Works.

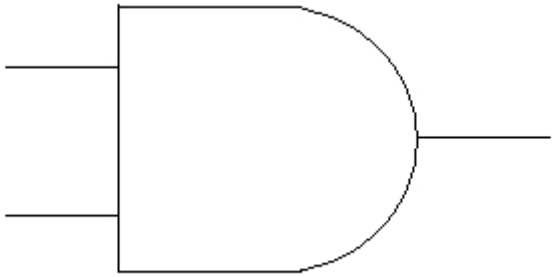
Basert på Wikipedia og programmet DigitalWorks ble det bestemt utseende til portene i Digital Circuit, se figur 5.5.

5.1.1 Bézier-kurver

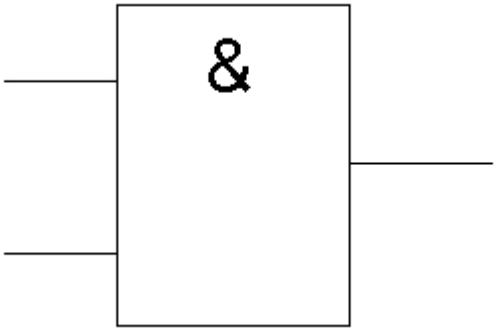
Generelt består portene av rette streker og kurver. En AND-port har en rund D-lignende krumning, mens en OR-port har to andre krumninger. Disse kurvene kan man tegne enten med en kvadratisk Bézier-kurve eller med en kubisk Bézier-kurve.

- Kvadratisk Bézier-kurve.

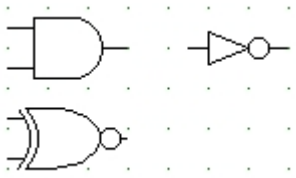
[1] Dette kapittelet er utarbeidet i samarbeid med min medstudent. Derfor benyttes «vi» formen.



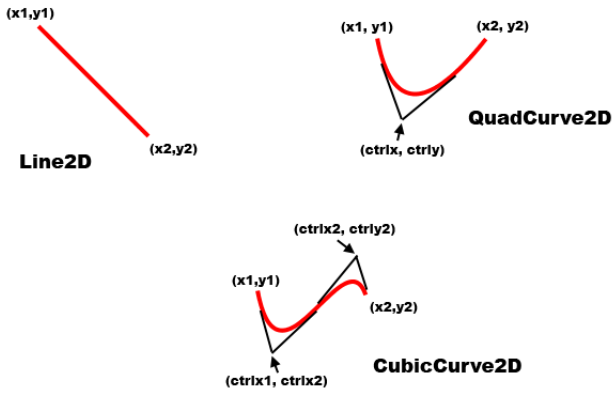
Figur 5.1: «Military AND»-symbol



Figur 5.2: «Rectangular AND»-symbol



Figur 5.3: Porter fra Digital Works



Figur 5.4: Linjer og kurver
(Bildet er hentet fra Sun[29].)

- Kubisk Bézier-kurve

Disse kurvene har begge et startpunkt og et sluttpunkt, og kurven kan styres ved hjelp av kontrollpunktene. Den vesentligste forskjellen mellom disse kurvene er at i en kubisk Bézier-kurve er det to kontrollpunkter, mens i kvadratisk Bézier-kurve er det ett kontrollpunkt. Kubisk Bézier-kurve kan også fungere som en kvadratisk Bézier kurve ved å sette like verdier for kontrollpunktene.

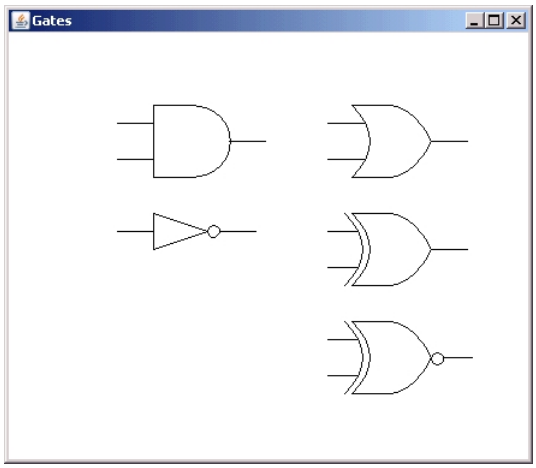
Hovedsakelig kan det brukes en kubiske Bézier-kurver til å tegne portene, men vi valgte likevel å bruke kvadratisk Bézier-kurve i tillegg. Grunnen til dette er at den som ser på koden skjønner med en gang at denne kurven kun har ett kontrollpunkt. En annen grunn som ikke har stor betydning er at det kan spares én peker ved å bruke kvadratisk Bézier-kurve.

Figur 5.4 forklarer mer om hvordan en linje, en kvadratisk Bézier-kurve og en kubisk Bézier-kurve ser ut.

5.2 Design av portene

I vårt første forsøk på å tegne porter laget vi en JPanel som fungerte som et koordinatsystem for de grafiske elementene som skulle fremvises på skjerm. Portene ble tegnet på panelet ved hjelp av Graphics2D sin draw-metode. I testfasen ble det brukt konstante verdier i koordinatsystemet.

I en OR-port skal den ytterste kurven være spissere enn i en AND-port. Ved å koble sammen to kubiske Bézier-kurver slik at de møttes på sluttpunktet, kan vi få en kurve som er spissere. For å tegne den innerste kurven i en OR-port ble det brukt en kvadratisk Bézier-kurve.



Figur 5.5: Ulike porter tegnet av Digital Circuit

5.2.1 NOT-port

NOT-porten bygges opp ved å sette sammen fem instanser av Line2D.Double: en for input, en for output og tre for den triangulære formen. Til slutt inneholder den en instans av Ellipse2D.Double. Figur 5.6 viser NOT-porten, sammen med koden for denne.

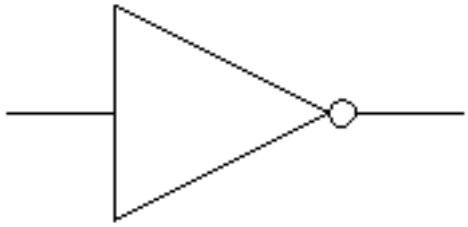
5.2.2 AND-port

AND-porten bygges opp ved å sette sammen seks instanser av Line2D.Double: to for input, en for output og tre for å danne en delvis rektangulær form. For resten av porten er det instansiert en bue av CubicCurve2D.Double. Figur 5.7 viser porten sammen med koden for denne.

5.2.3 XNOR-port

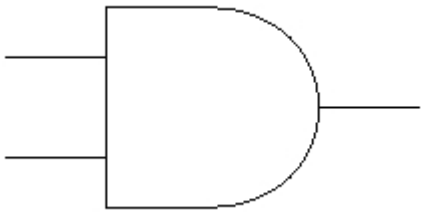
XNOR-porten er dannet med fem instanser av Line2D.Double; to for input, en for output, og en horisontal oppe og nede mellom hver kurve, se figur 5.8. Den består videre av to QuadCurve2D.Double, for å definere de to innerste kurvene til venstre. Kurven lengst til venstre indikerer at porten av typen XNOR.

For å få porten spiss er det tilsatt to CubicCurve2D.Double som går mot hverandre. Med CubicCurve2D.Double kan kurven bøyes på to steder for fin gradering. Til slutt er det satt inn en instans av Ellipse2D.Double for å indikere at denne porten er en inverter for XOR-porten. Figur 5.8 viser XNOR-porten sammen med koden.



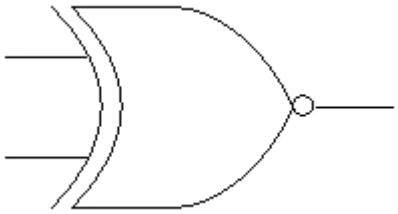
Figur 5.6: NOT-port

```
114         int s = 50; //størrelse
115         Line2D.Double line1_not = new Line2D.Double(s*3, s*5.5, s*4, s*5.5);
116         Line2D.Double line2_not = new Line2D.Double(s*4, s*5, s*4, s*6);
117         Line2D.Double line3_not = new Line2D.Double(s*4, s*5, s*5.5, s*5.5);
118         Line2D.Double line4_not = new Line2D.Double(s*4, s*6, s*5.5, s*5.5);
119         Shape circle2 = new Ellipse2D.Double(s*5.5, s*5.5-5, 10.0f, 10.0f);
120         Line2D.Double line5_not = new Line2D.Double(s*5.5+circle2.
121     getBounds().width, s*5.5, s*6.5+circle2.getBounds().width, s*5.5);
```



Figur 5.7: AND-port

```
56          int s = 50; //størrelse
57          Line2D.Double line1 = new Line2D.Double(s*3, s*2.5, s*4, s*2.5);
58          Line2D.Double line2 = new Line2D.Double(s*3, s*3.5, s*4, s*3.5);
59          Line2D.Double line3 = new Line2D.Double(s*4, s*2, s*4, s*4);
60
61          Line2D.Double line4 = new Line2D.Double(s*4, s*2, s*5, s*2);
62          Line2D.Double line5 = new Line2D.Double(s*4, s*4, s*5, s*4);
63
64      CubicCurve2D.Double curve =
65              new CubicCurve2D.Double(s*5, s*2, s*6.5, s*2,
66                  s*6.5,  s*4, s*5, s*4);
67
68      Line2D.Double line6 = new Line2D.Double(s*6.12, s*3, s*7.12, s*3);
```



Figur 5.8: XNOR-port

```
169         int s = 50; //størrelse
170         Line2D.Double line1_xnor = new Line2D.Double(s*8.85, s*8.5,
171                                     s*9.65, s*8.5);
172         Line2D.Double line2_xnor = new Line2D.Double(s*8.85, s*9.5,
173                                     s*9.65, s*9.5);
174
175         QuadCurve2D.Double curve_xnor = new QuadCurve2D.Double(s*9.5,
176                             s*8, s*10.5, s*9, s*9.5, s*10);
177         QuadCurve2D.Double curve2_xnor = new QuadCurve2D.Double(s*9.3,
178                             s*8, s*10.3, s*9, s*9.3, s*10);
179
180         Line2D.Double line3_xnor = new Line2D.Double(s*9.5, s*8,
181                                     s*10.5, s*8);
182         Line2D.Double line4_xnor = new Line2D.Double(s*9.5, s*10,
183                                     s*10.5, s*10);
184
185       CubicCurve2D.Double curve4_xnor =
186             new CubicCurve2D.Double(s*10.5, s*8, s*11, s*8,
187                 s*11.5,  s*8.5, s*11.7, s*9);
188       CubicCurve2D.Double curve5_xnor =
189             new CubicCurve2D.Double(s*10.5, s*10, s*11, s*10,
190                 s*11.5,  s*9.5, s*11.7, s*9);
191
192
193       Shape circle = new Ellipse2D.Double(s*11.7, s*8.89, 10.0f, 10.0f);
194       Line2D.Double line5_xnor = new Line2D.Double(s*11.95, s*9,s*12.7,s*9);
```

5.2.4 Konklusjon

For å tegne portene ble det benyttet instanser av Shape-objekter som for eksempel Line2D.Double, QuadCurve2D.Double og CubicCurve2D.Double. Deretter ble hver av disse instansene tegnet ved hjelp av Graphics2D sin draw-metode.

Det å tegne hver instans var en tungvint måte gjøre det på. Det ville også være vankeligere å finne ut om brukeren treffer en bestemt port med muspekeren eller ikke. Grunnet til dette er fordi det ikke er noen kobling mellom hver av de instanser som tegnes på panelet. Derfor ville ikke denne løsningen være aktuell for oss.

I dette kapittelet ble det avgjort om hva som trenges for å få utseendet til de forskjellige portene.

5.3 Portene med Shape-grensesnitt

De fleste av grafikk-objektene i Java er implementert med grensesnittet Shape. Dette ga inspirasjon til å gjøre det samme med våre grafiske elementer også. Ved å type-definere elementer som Shape-objekter, får man også bedre funksjonalitet, siden Shape-grensesnittet krever at hvert element skal implementere disse metodene:

- boolean contains(double x, double y)
- boolean contains(double x, double y, double w, double h)
- boolean contains(Point 2D p)
- boolean contains(Rectangle 2D r)
- Rectangle getBounds()
- Rectangle2D getBounds2D()
- PathIterator getPathIterator(AffineTransform at)
- PathIterator getPathIterator(AffineTransform at, double flatness)
- boolean intersects(double x, double y, double w, double h)
- boolean intersects(Rectangle2D r)

Java har en GeneralPath-klasse som kan brukes til å sette sammen Shape-objekter. GeneralPath-klassen kan også hjelpe oss til å koble sammen to Shape-objekter som møtes på et punkt. Ved å bruke dette kan man unngå å tegne hver instans av Shape-objekt. Lyttere kan også implementeres enklere for portene ved hjelp av contains-metoden til GeneralPath.

5.3.1 GeneralPath

GeneralPath er en klasse som kan brukes for å konstruere grafiske stier bestående av rette streker og kubiske og kvadratiske Bezier-kurver og kan inneholde flere del-stier. Viktige metoder fra denne klassen som kan brukes til å tegne er:

- moveTo(float x, float y)
- lineTo(float x, float y)
- quadTo(float x1, float y1, float x2, float y2)
- curveTo(float x1, float y1, float x2, float y2, float x3, float y3)
- closePath()

Implementering av metodene i Shape-grensesnittet vil være enklere dersom GeneralPath-klassen skal brukes, siden GeneralPath klassen er implementert med mange av de metodene som Shape-grensesnittet krever å implementere.

5.3.1.1 Metodene i GeneralPath

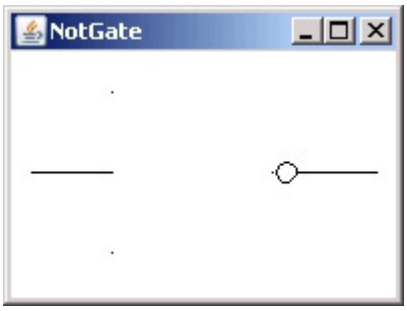
moveTo-metoden brukes til å flytte til startpunktet til der formen skal starte. lineTo-metoden brukes gjerne deretter for å trekke en linje fra det punktet man sist var på til det punktet man skal til. quadTo-metode kan konstruere en kvadratisk Bezier-kurve, curveTo-metode kan konstruere en kubisk Bezier-kurve og curveTo-metode trekker en linje fra det punktet en var på sist. Alle disse metodene er av typen void.

5.3.1.2 append-metoder i GeneralPath

GeneralPath har to forskjellige append-metoder:

1. append(PathIterator pi, boolean connect)
2. append(Shape s, boolean connect)

Metode nr.2 var mest nyttig siden den tok imot et Shape-argument. Andre argumentet til denne metoden skal være en «boolean» type som forteller om det innlagte objektet skulle kombineres med det neste. I forrige kapittelet er det allerede implementert porter med instanser av Shape-objekter, derfor velges det å bruke nr. 2 append-metoden til å se mulighete videre i dette kapittelet.



Figur 5.9: Punkter satt sammen i en triangulær formasjon

5.3.1.3 Nye måten for å tegne en port

- append(Shape s, boolean connect)

To linjer/punkter kan kobles sammen ved å angi andre parameter til append metoden med verdien «true».

append-metoden i GeneralPath kan brukes til implementere portene litt annerledes enn tidligere. Den nye muligheten som oppdages her er at det ikke trenges å opprette instanser av linjer. Isteden kan man bruke punkter og koble sammen til en linje. Med denne tanken klarte vi å implementere en annen løsningen til en NOT-port vist på figuren 5.9.

På figur 5.9 vises det hvor punktene satt i en triangulær formasjon, og ved å bruke append-metoden kobles disse punktene sammen til en bli en NOT-port. Figur 5.10 viser resultatet sammen med koden for dette.

5.3.2 GeneralPath tegner porten

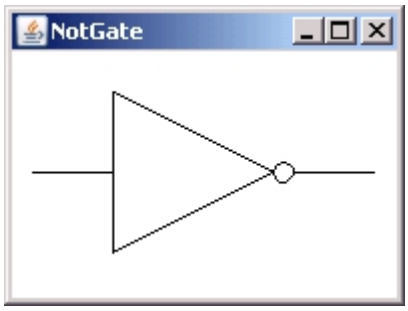
Med den forrige løsningen trenges det å gå gjennom to steg for å få tegnet en port. Først måtte det opprettes instanser av Line2D.Double og legge dem til GeneralPath etterpå. Dersom metodene fra GeneralPath benyttes til å gjøre hele jobben, unngår man to steger for dette. I tillegg vil GeneralPath redusere antall kodelinjer og gi bedre og mer oversiktlig struktur i koden.

Figur 5.11 viser en NOT-port sammen med koden, hvor GeneralPath og Ellipse2D.Double ble brukt til å tegne porten. Sirkelen som lages av Ellipse2D.Double blir tilføyd til GenerallPath ved å bruke append-metoden.

5.3.3 Konklusjon

Når det gjelder implementering av porter er det viktig å ha god struktur og ryddig kode. Dette gjør koden blant annet lettere å vedlikeholde og videreutvikle. GeneralPath tilfredsstiller kravene over og derfor velges det mest mulig å bruke GeneralPath til tegning av porter.

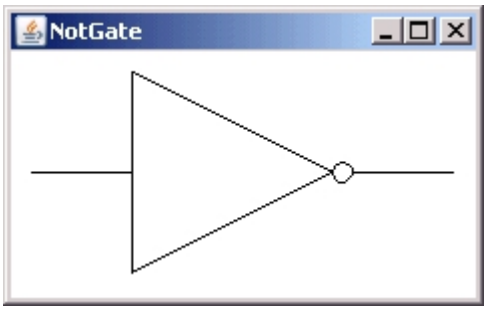
Når GeneralPath brukes til å tegne hele porten kan contains-metoden returnere informasjon og opplyse om det oppgitte punktet er innenfor



Figur 5.10: Figuren viser en hel form

Gjorde linjene om til punkter ved å sette like verdier for endepunktene:

```
13 public class ThirdMethod_NotGate implements Shape {
14
15   GeneralPath path;
16   Graphics2D g2;
17   float x, y;
18   int size;
19   public ThirdMethod_NotGate(int size, float x, float y) {
20     this.x = x;
21     this.y = y;
22     this.size = size;
23
24     Line2D.Double line1_not = new Line2D.Double
25     ((x - (size * 2)), (y),(x) - size, (y));
26     Line2D.Double line2_not = new Line2D.Double
27     ((x - size), (y) - size,(x - size), (y) - size);
28     Line2D.Double line3_not = new Line2D.Double
29     ((x + size), (y),(x + (size)), (y));
30     Line2D.Double line4_not = new Line2D.Double
31     ((x - size), size + (y),(x - (size)), size + (y));
32     Shape circle = new Ellipse2D.Double
33     ( x + (size), y - 5.0f, 10.0f, 10.0f);
34     Line2D.Double line5_not = new Line2D.Double
35     ((x + (size)) + circle.getBounds().width, (y),
36     (x + (size*2)) + circle.getBounds().width, (y));
37
38
39     path = new GeneralPath();
40     path.append(line2_not, false);
41     //Ved å bytte til true
42     path.append(line3_not, true);
43     //Ved å bytte til true
44     path.append(line4_not, true);
45     path.closePath();
46     path.append(line1_not, false);
47     path.append(circle, false);
48     path.append(line5_not, false);
49   }
```



Figur 5.11: GeneralPath brukt til å tegne NOT-porten

```
12 public class FinalMethod_NotGate implements Shape {
13
14   GeneralPath path;
15   Graphics2D g2;
16   float x, y;
17   int size;
18   public FinalMethod_NotGate(int size, float x, float y) {
19     this.x = x;
20     this.y = y;
21     this.size = size;
22     Shape circle = new Ellipse2D.Double
23     ( x + (size), y - 5.0f, 10.0f, 10.0f);
24
25     path = new GeneralPath();
26     path.moveTo((x - (size * 2)), (y));
27     path.lineTo((x) - size, (y));
28     path.closePath();
29     path.moveTo((x) - size, (y));
30     path.lineTo((x - size), (y) - size);
31     path.lineTo((x + (size)), (y));
32     path.lineTo((x - size), size + (y));
33     path.closePath();
34     path.append(circle, false);
35     path.moveTo((x + (size)) +
36       circle.getBounds().width, (y));
37     path.lineTo((x + (size*2)) +
38       circle.getBounds().width, (y));
39     path.closePath();
40   }
```

formen. På denne måten er det mye enklere å gjenkjenne hvilken port brukeren klikket på med musepekeren.

Kapittel 6

Som å lage et spill

Denne masteroppgaven har mange likhetstrekk med spillutvikling. I begynnelsen av prosjektet kom dette ikke så tydelig frem. Ettersom applikasjonen begynte å ta form og problemstillingen med å finne frem til den riktige porten en bruker velger blant alle portene på tegnebrettet kom det frem klare assosiasjoner til spillprogrammering og objektorientert programmering.

6.1 Generelt

Data og plattformbaserte spill kommer i mange former og ulike grensesnitt. Det finnes for eksempel tekstbaserte spill, puzzle-spill, 2D-spill, 3D-spill, osv. Spillutviklingsselskaper investerer flere titals millioner kroner i hvert prosjekt og det tar som oftest flere år å utvikle spillene. Et spill kan deles inn i fem basiske objekter[38]:

- Spillenhet: Det kan være hvilken som helst objekt i spillet. For eksempel spillerens avatar, en pistol, en hindring, et kjøretøy, osv.
- Spillenhetens form: Denne representerer formen til komponentene eller utseendet til et spillenhet.
- Spillhandling: Denne komponenten modellerer vanligvis spillmekanikken og diverse andre regler og betingelser som inngår i spillet.
- Spillstatus: Denne opprettholder og fører opp data gjeldende for en entitet eller dets form i spillet.
- Spillområdet: Den delen av spillet som representerer «verdenen/omgivelsen» i spillet. Det er her virkningene av handlingene utspiller seg.

Hovedfunksjonaliteten til et spill kan settes opp slik:

- Oppdatering av grafikkgrensesnitt.

- Mus / tastatur / bruker - Input lytter.
- AI-kontroller (Artificial Intelligence controller).

Det er flere sammenlikninger som kan gjøres mellom denne oppgaven og et dataspill. Vi har et panel hvor elementene skal kunne pekes på, dras og flyttes rundt. Dette skal brukeren kunne gjøre med både mus og tastatur. For å få til dette burde elementene på panelet kunne lytte på brukerens itreageringer. Hvert element skal kunne koples til ulike elementer og må følge spesifiserte regler og betingelser for å gjøre dette. For eksempel kan to logiske porter koples sammen for å lage en annen port med en annen funksjonalitet, men utgangen fra den ene porten skal ikke kunne koples til utgangen til den andre. Som sagt må brukeren følge de regler og betingelser som er satt av programmereren. Applikasjonen skal bygges ut slik at brukeren blir varslet dersom forutsetningene for riktig programbruk brytes. Til slutt må programmet også kunne simulere kretsen.

En sammenlikning av vår applikasjon og de fem basiske objektene som utgjør et spilloppsett, se listen over, viser at spillenhet samsvarer med porter og andre elementer som flip-flop osv. Spillenhetens form vil beskrive formen og utseendet til elementene i programmet. Disse elementene vil ikke forandre seg ofte, men endre form dersom brukeren endrer innstillingene for størrelse eller ved for eksempel å rotere portene med tastatur og mus. Spillhandling vil utgjøre sammensetningen av regler og betingelser i programmet. Spillstatus vil i vårt program utgjøre en strukturert oppsamling av data gjeldende for hvert element til enhver tid. Dette kan for eksempel være hvilken tilstand en port er i, hvor den er plassert, hvilken andre enheter den er koplet til, hvilken input/output som er ukoplet og ledige osv. Spillområdet sammenlignet med vårt område er panelet der brukeren vil tegne kretser.

Animasjon er ganske sentralt i et spill men til oppgaven vår vil ikke være nødvendig med mye animering. Animering vil komme bare når brukeren gir strøm og vi skal lede strømmen gjennom kretsen og til å animere «LED».

Hovedforskjellen mellom vår applikasjon og et dataspill er i hovedsak at vi ikke har 3D-grafikk og AI-kontroller. En AI-kontroller er en smart algoritme som skal etterligne intelligens og frambringe dette i spillelementer styrt av datamaskinen. I vårt program er disse ikke nødvendige. Hovedfokus vil være på lage et grafisk designverktøy hvor funksjonalitet tilknyttet mus og tastatur vektlegges. Brukergrensesnittet utvikles med tanke på brukervennlighet.

6.2 Hvorfor Java i spillprogrammering?

Det er en generell aksept blant store spillutviklingsstudioer at spill må programmeres i C, C++, assembler eller andre programmeringsspråk enn Java. Grunnene de oppgir til dette er mange[35]:

- Java er for treg til spillprogrammering.

- Java har minnelekkasje.
- Java er for høynivå.
- Java applikasjonsinstallasjon er et mareritt.
- Java støtter ikke spillkonsoller.
- Ingen bruker Java til å skrive et «virkelig» spill.
- Sun Microsystems er ikke interessert i å støtte Java-spill.

Boka «Killer Game Programming» snakker om forholdet mellom Java og spill i første kapittel. Her påvises det at alle punktene ovenfor egentlig ikke stemmer. Det nevnes at Java omtrent er like raskt som C++. Man unngår vanligvis minnelekkasje i andre programmeringsspråk ved å benytte spesielle teknikker. Det er en ren misforståelse om minnelekkasje i Java. I Java forekommer det ikke minnelekkasje siden «garbage collector» håndtere dette automatisk for programmereren. Det stemmer at Java er et høynivåspråk men det finnes ingen begrensninger for direkte tilgang til «Graphics hardware» og andre eksterne enheter. Installasjonen av Java er ingen mareritt dersom det benyttes en brukervennlig installasjonspakke. Det kommer stadig flere utmerkede Java-spill og Sun tilbyr enorm støtte gjennom sin egen hjemmeside men også via andre sider sponset av dem.

Blant de beste argumentene for ikke å benytte Java til spillutvikling har vært hastighet. At spill laget i Java ikke er optimale med tanke på hastighet stemte kanskje i 1996 da denne type kritikk dukket opp. Sun har med hver nye utgivelse av Java økt hastigheten betydelig. Den første versjonen av Java, JDK 1.0, var 20 til 40 ganger tregere enn C++. J2SE 5.0 som er den siste versjonen er kun 1.1 ganger så treg[35]. Dette kommer også frem i prosessbytte eksperimentet jeg nevner i kapittelet om tråder. Disse tallene er nok relative, og avhenger svært mye av hvordan man programmerer. Men vi ser en tydelig trend blant utviklere at Java brukes i større grad nå enn før.

Siden Swingkomponenter er laget og kontrollert av Java med lite OS-støtte hevdes det at disse er trege. En grunn til denne påstanden kan være at det finnes et ekstra lag ovenfor OS for prosessering. Kanskje det er mer riktig å si at «programmet» er tregt og ikke Swing, Swing i seg selv er ikke noe stort. Det er først når man begynner å legge inn mye kode i «events handler» at GUI vil bli treg. En bra artikkel om dette er skrevet av Sermet Yucel[50].

Forbedringer i kompilatordesign som «just-in-time» kompilering analyserer kode som brukes mange ganger og kompilere denne aggressivt. Dette har gjort Java mye raskere og mer populært for spillutvikling. Spesielt merkes denne trenden innen spill for mobilmarkedet hvor Java har den største

andelen. Flere av spillene utviklet i Java har mottatt en rekke priser og blitt bestselgere.

J2SE 5.0 inkluderer OpenGL-basert pipeline for Java2D. Bruken av «pipeline» gir en økt fartsøkning for enkle renderingsfunksjoner for tekst, bilder, linjer, og fylte former, men også til mer kompliserte omforminger og malinger[31]. Siden denne oppgaven skal utvikles i Java med bruk av Java2D-pakken, og basert på de faktaene beskrevet over, finner jeg ingen innvendinger i å benytte Java som utviklingsspråk. Erfaring fra prosjektet viser også klart at Java er raskt nok til formålet.

6.3 Sprites

En sprite er et to- eller tredimensjonalt bilde eller animasjon som er integrert inn i en større scene. Sprite kan også være et grafisk objekt som for eksempel representerer spilleren ved å reagere på tastetrykk og musehendelser.

Sprites ble opprinnelig oppfunnet som en metode for raskt å kunne sammen sette flere bilder i todimensjonale spill ved å bruke spesialdesignet hardware. Ettersom dataytelsen forbedret seg ble denne type optimalisering unødvendig. Termen «sprite» utviklet seg etter dette til å referere spesielt til de todimensjonale bildene som ble integrert inn i en scene. Figurer som nå enten ble generert av tilpasset hardware eller bare av software ble alle kalt sprites. Da tredimensjonal grafikk ble mer rådende, ble utrykket brukt til å beskrive en teknikk hvor flate bilder blir integrert sammen til å danne mer kompliserte tredimensjonale scener[8].

6.3.1 Hardware sprites

I tidligere dataspill var sprites en metode for å integrere ubeslektede bitmaps. På denne måten kunne disse bitmapene vises som en enkel bitmap på skjermen.

Blitter er en hardware-implementering av «Painter's algorithm» som refererer til en enfoldig maler som først maler fjerntliggende objekter i en scene for deretter å dekke de samme objektene med nærliggende objekter. For hver enkel sprites-frame blir en bit-blokk overført til en rask, stor og kostbar frame-buffer, for deretter å bli sendt til skjermen. Denne blitteren har i senere tid blitt renavnet til «graphics accelerators», men forekommer nå som en mer komplisert rendringsalgoritme[8].

6.3.2 Sprite-klasse

Generelt en Sprite implementeres av en klasse som inneholder:

- Posisjon
- Hastighet

- Inkrementering for oppdatering.
- Metoder for å laste inn et bilde til en sprite og spille av en sekvens av bilder for animasjon.

Resten er ikke likt for alle sprite-objekter og må kodes spesifikt. Et konkret sprite-objekt kan inneholde metode for å oppdatere spriten. Et slikt objekt håndterer bruker-interageringer, oppdagelser av kollisjon med andre sprites men også hindringer, respons og lydeffekter fra disse.

6.3.3 Konklusjon

Vi fant ut at det var best å tegne elementene ved hjelp av GeneralPath og ikke bruke bilder i forbindelse med sprites. Å bruke bilder for å tegne elementene ville gjort kobling av elementer vanskeligere og skalering av elementene ville ha ført til redusert kvalitet på tegningene. Men Sprite kan være nyttig til å animere simuleringen.

Kapittel 7

Oppbygging av Digital Circuit

Å designe og bygge opp strukturen i et program på en gjennomtenkt måte kan være en avgjørende faktor for å skape et velfungerende program. God struktur kan føre til programmer som er enklere å videreutvikle, enklere å feilsøke og har færre feil.

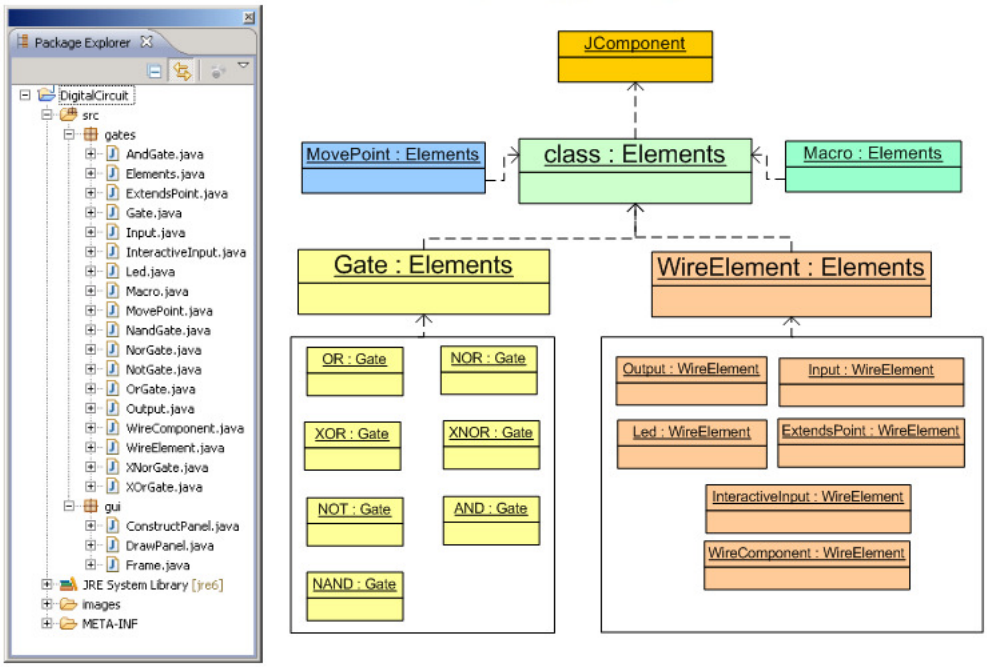
7.1 Klassestrukturen i Digital Circuit

Klassene som er valgt, gjenspeiler de komponentene som eksisterer i virkeligheten. Modellen vist på figur 7.1 er en forenklet abstraksjon av virkeligheten. Den hjelper programutvikleren til å håndtere kompleksiteten som kan ligge i oppgaven. Denne modellen er en overordnet modell og kan etter behov splittes i mindre deler med mer detaljert informasjon. Alle komponenter i Digital Circuit er objekter av subklassen Elements og alle disse klassene er skrevet av meg. Elements-klassen arver selv «JComponent»-klassen.

Gate og WireElement er to hovedkomponenter av Elements-klassen. Gate er superklassen for alle digitale porter og WireElement er superklassen for alle kabelelementer i programmet.

MovePoint er en selvstendig komponent som styrer bevegelser og flytting av ledninger. Dette elementet er en hjelpekomponent som arver Elements-klassen. Vi har også makroer i programmet, som bygges opp ved hjelp av Macro-klassen og arver Elements-klassen.

I pakkeoversiktsbildet vises hvilke pakker som inneholder de ulike klassene. Gates-pakken inneholder klasser som representerer de ulike elektroniske komponentene brukt i programmet. Under GUI-pakken ligger de klasser som tilhører brukergrensesnittet.



Figur 7.1: Pakkeoversikt og Modell -diagram

7.2 Oppretting av egne grafiske komponenter i Java

Java gir programutviklerne anledning til å definere egne grafiske komponenter ved å arve klassen Component eller JComponent. Det er nødvendig å avgjøre hvilken av de to klassene som vil være mest hensiktsmessig til vårt formål?

Component er en abstrakt superklasse for ikke-meny-relaterte AWT-komponenter. Et objekt av denne typen vil inneholde den grafiske framstillingen av det som vises på skjermen, og kan i tillegg inneholde lyttere som muliggjør at komponenter reagerer på brukerens handlinger. De mest kjente subklasser til Component er Label, List, Scrollbar og Container. Denne klassen kan også utvides til å opprette lettvektskomponenter[3]. Mer om AWT, se kapittel 3.1.1.

JComponent klassen er basisklassen for både standard og egendefinerte Swing- komponenter. Unntaket er «Top-level containers»[4] som ikke er en subklasse av JComponent. De fleste klasser hvis navn starter med bokstaven J vil være en Swing-komponent. JComponent selv arver fra beholderen (Container) som videre arver fra Component.

For at egendefinerte komponenter skal arve Swing-arkitekturen til Java må de være subklasse av JComponent[4]. Dette programmet vil være enklere å bygge opp dersom layout funksjonaliteten er innebygd i basisklassen. Basisklassen er den klassen som våre elementer skal arve fra. Layout gir

oss mulighet til å sette sammen flere komponenter i én komponent med en enkel funksjon som add. På grunn av fordelene og enkelheten som layout-funksjonaliteten tilbyr, velger jeg å implementere elementene ved hjelp av JComponent. Vi kan ikke bruke Component-klassen siden denne ikke støtter layout-funksjonaliteten.

JComponent-klassen brukes med null layout til å bygge opp komponenter i Digital Circuit. Med null layout menes at det ikke brukes noen oppsetthåndterere (layout managers) i Java. Elementene plasseres i stedet ved å oppgi x og y punkter. Mer om de forskjellige «layout managers», se kapittel 3.2.4.2.

7.3 Oppbygging av porter

I JComponent objektet tegnes alle komponenter ved hjelp av GeneralPath. Tegningene av porter og andre komponenter i programmet er todimensjonale. Disse komponentene må reagere på tastetrykk og musbevegelser. Vi oppnår disse kravene ved å implementere porter med flere Listener-grensesnittet.

I tillegg må portene inneholde disse egenskaper:

- Posisjon x,y
- Størrelse
- Antall input
- Utgang
- Utgangsignal
- Data struktur (neste og forrige peker)
- Init-metode
- om porten er valgt (Selected)
- Simulerings() metode
- Sendsignal() metode

7.3.1 Historien bak oppbygging av portene

I tidlig fase av oppgaven ble portene implementert ved hjelp av Shape-grensesnittet, se kapittel 5.3. Når en ny port skal tegnes på tegnepanelet med dette grensesnittet, må det legges til i listen som brukes ved tegning av komponenter. Ved hjelp av Graphics2D sin draw-metode tegnes elementene som ligger i listen på panelet. Det vil si at hver gang en legger til et nytt element på panelet, kalles repaint-metoden. Dette medfører unødvendig

tidsbruk og ressursbruk, fordi alle tidligere elementer i listen gjennomgås og tegnes på nytt.

Objekter som er av typen Shape brukes til å tegne, vi kan gjøre dette ved hjelp av tegnefunksjoner i Java. Det kan ikke legges til en lytter til et slikt objekt, fordi de vil fungere bare som en tegning og ikke som et objekt som kan lyttes til. For å ha lytterfunksjonalitet i et objekt, må klassen være en subklasse av Component, JComponent eller de som arver disse. Dette har ført til at det ble utviklet en Elements-klasse som arver fra JComponent, og som er superklasse for alle elementer i Digital Circuit.

7.3.2 Implementering av Gate-klassen

Hver enkel port i Digital Circuit er et Gate-objekt. Dette objektet inneholder informasjon som er felles for portene, se underkapittel 7.3.

Gate er en abstrakt superklasse som arver fra Elements og implementerer følgende grensesnitt:

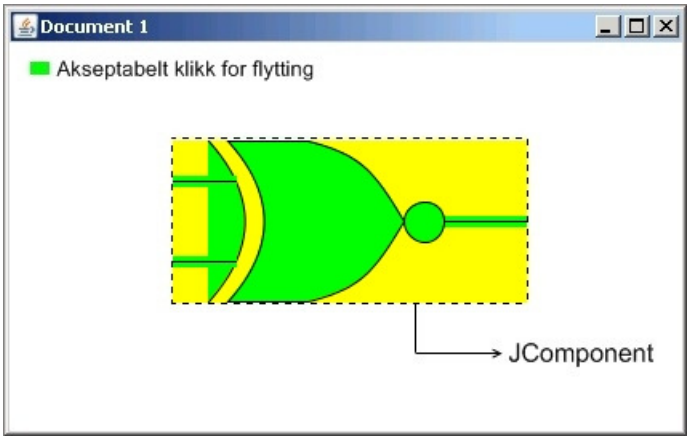
- MouseListener
- MouseMotionListener
- KeyListener

Gate-klassen implementerer noen av de metodene listet under. Disse metodene overskriver de i JComponent:

```
// overrides Component
public Rectangle getBounds();

// overrides JComponent
public void paintComponent(Graphics g);

// overrides JComponent
public boolean contains(int input_x, int input_y);

// overrides Component
public Dimension getSize();
```

Gate-klassen inneholder også abstrakte metoder som etterkommere av denne klassen må implementere:

```
public abstract void startSimulering();

public abstract void sendSignal();

public abstract void create();
```



Figur 7.2: Formen til XNOR -porten

Portene er definert med ulik form/utseende og ligger nær opp til konvensjonen for hvordan porter og andre komponenter skal se ut, se kapittel 5. Siden portene arver fra JComponent-klassen vil formen normalt være som en rektangel, se figur 7.2. Nesten alle kjente komponenter i Java har en rektangulær form, for eksempel JLabel og JButton.

JComponent-klassen i Java inneholder en spesiell metode som kan brukes til å definere formen til ikke-rektangulære komponenter. Metoden heter contains(int x, int y). Contains-metoden krever to parametrer x og y, som returnerer sann hvis det oppgitte punktet utgjør en del av formen. I Digital Circuit er denne metoden implementert til å sjekke opp mot contains-metoden til GeneralPath. Siden GeneralPath brukes til å tegne, blir ikke arbeidet med å definere formen på portene en krevende oppgave.

Contains-metoden definerer det grønne område på figur 7.2, og som utgjør den lovlige delen av formen. Porten kan kun flyttes dersom brukeren trykker innenfor det grønne området.

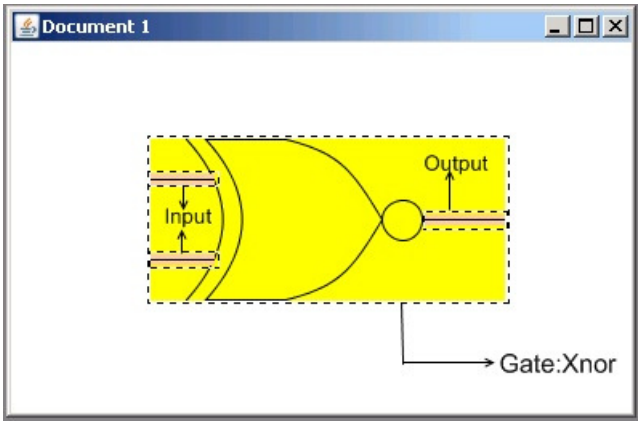
7.3.3 Implementering av en enkelt port

I Gate-klassen implementeres de felles metoder og variabler som skal gjelde for den enkelte port. I tillegg vil hver port få sin egen mus- og tastatur-lytter. Figur 7.3 viser hvordan en XNOR-port er bygget opp. Legg spesielt merke til inngangene og utgangen på porten.

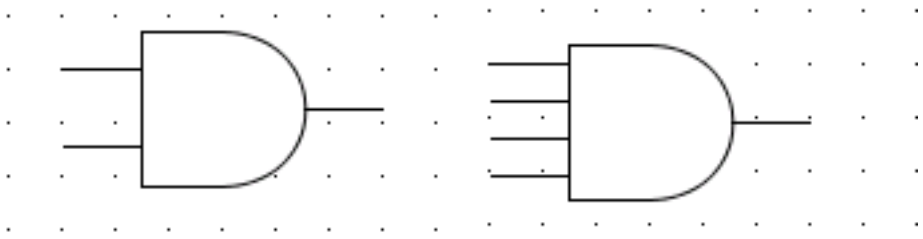
I XNOR-porten som vist i figur 7.3 er det to innganger av typen Input og en utgang av typen Output. Formen til selve porten er tegnet av «GeneralPath».

7.3.4 Porter med flere innganger

```
public XNorGate(int størrelse, int antallInp) {
```



Figur 7.3: Oppbygging av XNOR -port



Figur 7.4: En AND-port med flere innganger i Digital Circuit

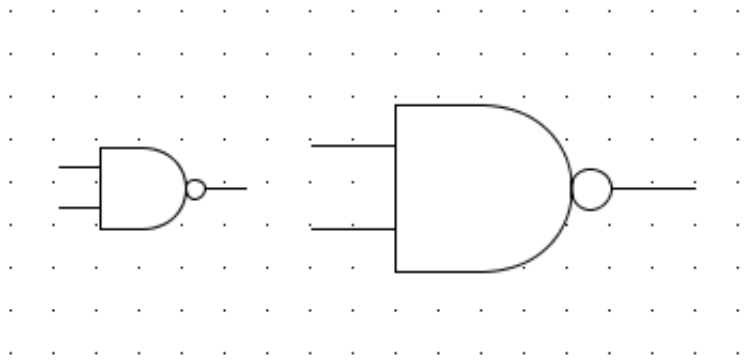
Antall innganger for en enkelt port kan variere fra port til port. Derfor er portene implementert med tanke på at en kan bestemme størrelsen og antall innganger. Konstruktøren til den enkelte port krever to parametrer som er av typen int. Den første parameteren forteller størrelsen og den andre parameteren angir antall innganger porten skal ha. På figur 7.4 vises en AND-port med 2 og 4 innganger.

De fleste studenter vil ikke ha behov for porter med mange innganger, og på grunn av begrenset tid for denne oppgaven er maks antall innganger en port kan ha er begrenset til 4. Med en fast grense vil implementeringen og det visuelle fremstillingen av en port ha klare fordeler.

Per i dag har ikke brukeren mulighet til å bestemme selv antall innganger. Ved videreutvikling av programmet kan en implementere funksjonalitet som kan gi brukeren mulighet til å bestemme antall innganger på portene.

7.3.5 Skalering av porter

Portene er bygget opp slik at brukeren selv kan bestemme størrelsen. Når brukeren utfører en høyre museklikk på porten, vil de få mulighet til å zoom



Figur 7.5: Zooming av NAND-port

in og ut. Figur 7.5 viser en skalering av NAND-porten.

I programmet er det bestemt slik at hver gang brukeren forstørrer porten økes med en konstant på fem, og ved forminsking minkes med en konstant på fem i størrelsen. Forstørring og forminsking gjøres ved at porten tegnes på nytt med den nye størrelsen hvergang.

Portene ble definert og implementert med en størrelse under oppbygging. Ved å oppdatere den definerte størrelsen, og ved hjelp av metoden som tegner porten ble skalering muliggjort. I tillegg bestod portene med flere deler som inngang og utgang. På grunn av dette benyttes det ikke klassene fra Javas standardklasse bibilioteket til forstørring og forminsking, selv om Java2D har innebygde mekanismer for å produsere vektorbasert-grafikk[47].

7.3.6 Oppbygging av Input og Output

Innganger og utganger er ikke en del av tegningen til porten, men er bygget opp som komponenter av typen WireElement. De skal kunne lytte til brukerhendelser og skal kunne fungere etter noen predefinerte regler. Disse elementene skal være en del av de interaktive objektene i programmet, og kan derfor ikke være en del av tegningen for porten.

Det er ikke enkelt å treffe en linje med musmarkøren. Derfor er Input og Output komponentene utformet til å være litt større enn bare selve linjen, se figur 7.3. Dersom det viser seg at det stadig er vanskelig å treffe linjen, kan denne komponenten utvides. Dette bør også være en del av videreutviklingsarbeidet.

7.3.7 Alternativ løsning

En alternativ løsning kunne ha vært å bruke bilder, istedenfor å tegne portene med Graphics2D. Ved å bruke bilder vil vi få et dårligere skaleringsresultat. En annen ulempe er at `contains`-metoden må redefineres for å bestemme hvilken del av bildet som utgjør porten, noe som er betraktelig mer krevende.

7.3.8 Konklusjon

Implementering av portene med Shape-grensesnittet som gjort i tidligere fasen er forkastet. Portene ble senere implementert med Elements-klasse fom arver JComponent. Dette viste seg å være enkelt og dekker kravene som stilles til en port.

Ved å bruke Graphics2D og dens innebygde funksjonalitet som en løsning får å tegne portene, utføres skaleringen uten tap av kvalitet. Innganger og utganger som egne komponenter har ført til enklere og mer effektiv implementering av avanserte funksjoner.

7.4 Oppbygging av ledningssegmenter

Dette underkapittelet tar for seg hvilke muligheter, ideer og problemer som underveis dukket opp ved implementasjon av ledningssegmenter. Tanken har vært å ikke begrense seg til kunnskap som jeg allerede innehar, men også kunne greie å hente erfaring fra andre type problemer og få et bredere perspektiv på oppgaven.

7.4.1 Historien bak oppbygging av ledningskomponenter

I en tidligere fase av oppgaven ble ledninger tegnet som linjer ved å bruke GeneralPath på selve tegnepanelet. Dette ble gjort for enkelhetsskyld men også for å utforske mulighetene som kunne ligge tilstede. Tegnepanelet kalles i programmet for DrawPanel som arver fra JPanel. På dette panelet utføres all oppbygning av digitale kretser. Modningsprosessen ble avsluttet med den erfaring at tegninger ikke kan implementeres med lyttere.

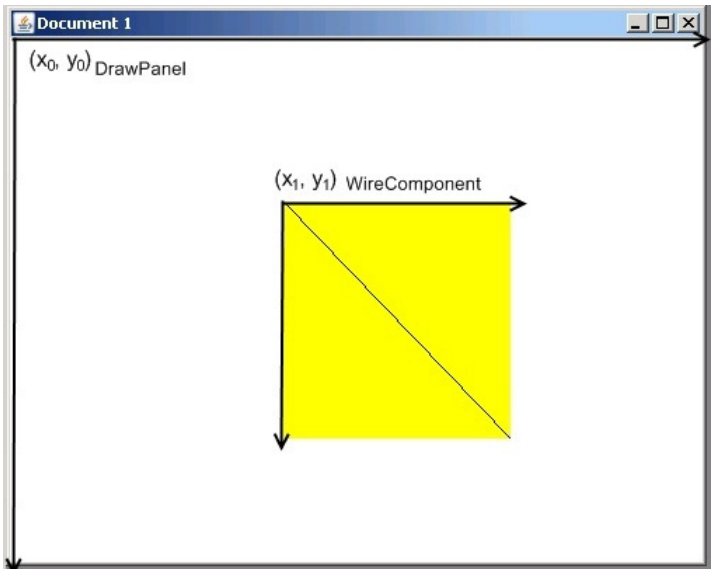
7.4.2 Alternative løsninger

I denne delen av underkapittelet skal to alternative løsninger for hvordan ledningssegmenter kan implementeres. Vi vil se litt på hvilken av løsningene som vil være meste aktuell til vårt formål.

7.4.2.1 Krav til ledningssegmenter

Hver ledning er bygget opp av én eller flere rette ledningssegmenter. Disse ledningssegmentene skal ha en stor grad av interaktivitet. Dermed er det først og fremst viktig å implementere disse komponentene slik at brukeren enkelt kan treffe dem.

Et problem kan oppstå når brukeren ønsker å marker en linje med musmarkøren. Det kan tenkes at tykkelsen på linjen er så smal at brukeren vil ha vankeligheter med å treffe den. Vi ønsker derfor å lage en slik funksjonalitet at bruker kun trenger å treffe nær nok et ledningssegment.



Figur 7.6: Oppbyggning av WireComponent

7.4.2.2 Alternativ løsning 1

Når brukeren trykker første gang på tegnepanelet med musmarkøren lagres startpunktet for ledningssegment, og ved andre gang trykk lagre sluttpunktet. Videre brukes disse to puktene til å opprette et ledningssegment av typen WireElement. Disse punktene er angitt i forhold til tegnepanelet siden det er her brukeren trykker.

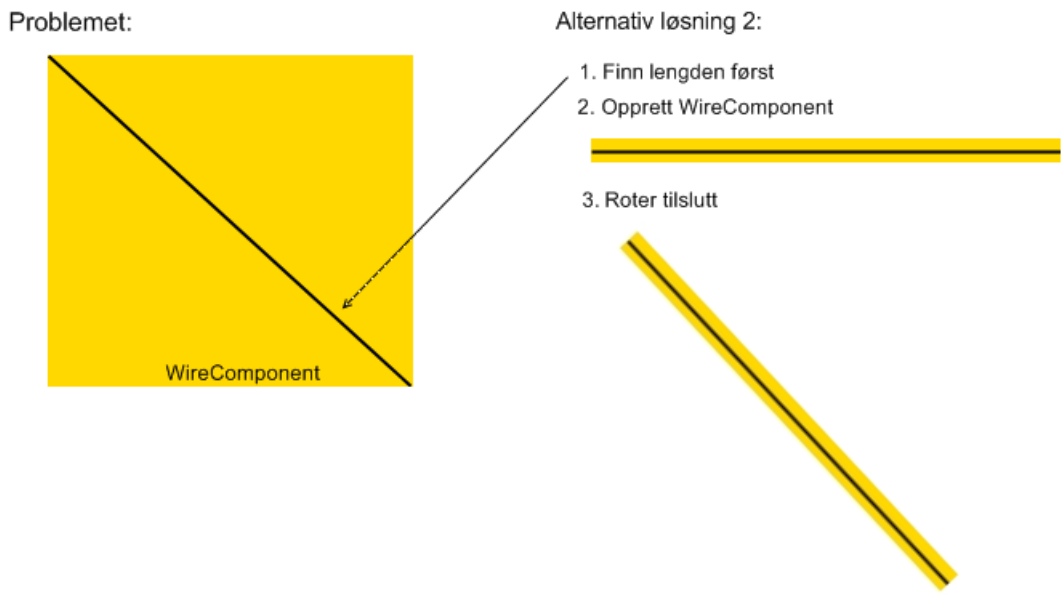
Det er viktig å huske at i den nye lednigskomponenten vil det ikke være samme x- og y-akse som i tegnepanelet, se figur 7.6. Derfor er det viktig å konvertere disse punktene i forhold til ledningssegmentet.

Ved å bruke de konverterte punktene kan linjen i et nytt ledningskomponent tegnes og deretter kan denne komponenten plasseres på tegnepanelet.

Dette forslaget krever redefinering av contains-metoden.

7.4.2.3 Problemet med alternativ 1

Dersom ledningselementer implementeres som egne komponenter av typen WireElement i oppgaven, vil komponentene normalt ha en rektangulær form som nevnt tidligere. Størrelsen vil da være bestemt av stigningstallet til linjen, jo mer linjen skråner, jo større blir komponentens areal. Ledningskomponenten vil derfor inneholde uaktuelle deler, og en redefinering av contains-metoden er nødvendig for at dette skal fungere.



Figur 7.7: Illustrasjon av ideen

7.4.2.4 Alternativ løsning 2

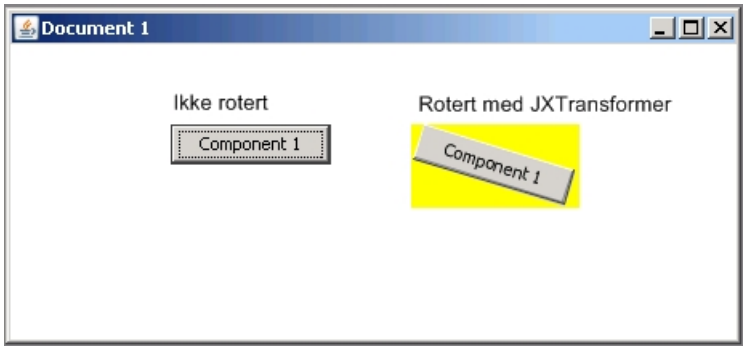
Dersom ledningskomponenter bare inneholdt lovlige deler, ville det ikke ha vært nødvendig å redefinere contains-metoden. Det ville i tillegg ha vært enklere å legge til lyttere.

Figuren 7.7 viser ideen for å løse problemet. Ideen er at man først finner lengden av ledningen som skal tegnes, og deretter tegner en vannrett linje i den nye ledningskomponenten. Når linjen er vannrett, vil ikke størrelsen til komponenten være stor fordi stigningstallet da vil være lik null. Når det er opprettet en ledningskomponent med en vannrett linje, må komponenten roteres slik at linjen peker mot riktig tilhørende komponent fra begge sider, se figur 7.7.

For at denne ideen om rotasjon skal fungere og for å unngå redefinering av contains-metoden, må komponenten beholde samme form etter rotasjon, slik vist i figur 7.7. Hvis ikke må roteringsmetodikken inneholde metoder for å transformere lyttere.

7.4.2.5 Mekanismer i Java for rotasjon

Det eksisterer ingen metoder i JComponent for rotering som kan brukes i implementeringen av alternativløsning nr. 2. Men det finnes noen hjelpeklasser som kan støtte implementeringen av denne funksjonaliteten. Derimot eksisterer det ingen mekanismer i Javas standard klassebiblioteket som kan benyttes direkte for å få utført en slik rotasjon av Swing/JComponent-objekter.



Figur 7.8: Rotert med JXTransformer

7.4.2.6 En ekstern klasse for rotasjon

En løsning for å rotere JComponent/Swing-komponenter finnes på nettstedet weblogs.java.net[45]. Denne løsningen er implementert av Alexander Potochkin og heter JXTransformer. Alexander Potochkin sier at han brukte nesten 3 måneder på å finne den generelle løsningen for rotasjon av Swing-komponenter.

JXTransformer roterer komponentene ved å bruke «AffineTransform»-klassen sammen med Graphics2Ds innebygde funksjoner. « AffineTransform»-klassen er tilgjengelig i Javas standard klassebibliotek og hjelper oss med oversetting av koordinatpunkter til rotasjon, forstørring og skyving av komponenter.

En komponent som roteres ved hjelp av JXTransformer, vil ikke beholde samme form etter rotasjon, se figur 7.8. Det gule området rundt komponenten viser at det alltid vil ha en rektangulær form. Siden JXTransformer også transformerer lyttere, vil ikke det gule området rundt komponenten være interaktivt. Det vil si at denne måten å transformere på tilsvarer å redefinere contains-metoden.

7.4.2.7 Konklusjon

Det eksisterer ingen mekanismer i Java for å løse alternativ nr. 2. Løsningen til Alexander Potochkin er unødvendig komplisert til denne oppgaven, selv om løsningen hans er veldig god. Dermed vil alternativ 1 om redefinering av contains-metoden være best egnet til vårt formål.

7.4.3 Valg av datastruktur

Datastruktur er en systematisk måte å organisere og strukturere data på i en datamaskin. Effektiv organisering av store mengder med data ved hjelp av algoritmer, muligjør at operasjoner og beregninger på disse kan utføres på en mer optimalisert og raskere måte. Valg av riktig datastruktur vil være

avhengig av hvilke faktorer som skal prioriteres framfor andre. Faktorer som kan være avgjørende for hvordan et program oppleves, vil være avhengig av om datastrukturen er laget med tanke på tidsbesparelse eller for å unngå mest mulig diskplass. Fornuftig og gjennomtenkt datastruktur kan minimere antall sammenligninger i et program. Bruken av minneplass vil minimaliseres og minst mulig diskplass vil bli benyttet[12].

I mange programmeringsspråk eksisterer det ferdige moduler i språkets standardbibliotek som programmereren kan anvende for å implementere effektive algoritmer. Java har «java.util»-pakken som inneholder ulike samlinger av datastrukturer. Datastrukturer kan deles i to hovedgrupper; lineære datastrukturer og datastrukturer som baserer seg på grafer. Lineære datastrukturer blir nærmere beskrevet i neste underkapittel.

Et godt gjennomtenkt valg av datastruktur har vært helt nødvendig for at elementene skal kunne henge sammen, og i tillegg at det skal være mulig å kunne simulere disse på en effektiv måte. Det er også behov for lister, dette for lagring av midlertidige elementer under tegning av en krets. Det ble konstatert at det var behov for en lineær datastruktur, og under denne strukturen er det også flere valg. Vi skal nå drøfte hvilke lineære datastrukturer som vil være mest hensiktsmessig for å bruk i denne oppgaven.

7.4.3.1 Array, ArrayList, Vector, LinkedList og HashMap

De viktigste lineære datastrukturene er lister, herunder linkede lister, «arrayer», hashtabeller, stakker og køer. Binære søketrær og heaper er de kjente datastrukturer som baserer seg på grafteori.

I et vanlig array må en bestemme størrelsen på forhånd, altså hvor mange elementer som skal kunne lagres. I tillegg også hvilken type, altså om den skal være av typen String, int eller en anne type objekt. Et vanlig array er raskere enn ArrayList, og anbefalt å bruke dersom en vet størrelsen på forhånd[39].

ArrayList og Vector er nesten av samme type lineær datastruktur, unntatt at Vector er synkronisert på den måten at to eller flere tråder kan utføre operasjoner i denne listen. Vector må synkronisere jobben og vil derfor være mye tregere enn ArrayList. Dersom det benyttes ArrayList må synkroniseringen utføres ved hjelp av hjelpemetoder eller klasser. Internt er disse to typene implementert som en vanlig array. Når det legges inn et nytt element i disse typer lister, vil den lage en nytt array med størrelsen n+1. Alle elementene vil bli kopiert over til den nye arrayen. Det nye elementet vil da bli plassert på indeks n+1. Det er også mulig å unngå omfordelingen ved å bestemme en størrelse på forhånd, dette kan gjøres med metoden ensureCapacity(int requestCapacity). Når den bestemte størrelsen brukes opp, vil omfordelingen startes igjen.

«LinkedList» er sekvens av noder med dobbeltlenket liste, og bruker ikke et vanlig array internt. Hver node i denne listen inneholder to pekere, en til forrige node og en til neste. Elementet kan plasseres hvor som helst i listen

på kortere tid uavhenging av om plasseringen er foran eller bak. Det vil si at innlegging og fjerning av elementer går mye raskere enn de andre listene. LinkedList implementerer kø-grensesnittet, og kan dermed brukes som en kø dersom det er behov for dette.

For HashMap brukes det en nøkkel for å legge inn et element. Dermed trenges det ikke å søke gjennom som et vanlig array for å finne et bestemt element. Søking og sletting av elementer er derfor veldig effektivt i en hashmap. Når det finnes en unik nøkkel for det elementet som skal legges inn, viser min erfariong at hashmap vil være det beste.

7.4.3.2 Konklusjon for valg av datastruktur

Listestrukturer brukes til å lagre og holde rede på elementene i en bestemt rekkefølge. I denne applikasjonen er alle komponenter/elementer som brukes visuelle, hvor brukeren selv bestemmer rekkefølgen av disse komponentene under tegning av en krets. Når brukeren velger tilfeldig av de komponentene som ligger på tegnepanelet, må programmet finne hvilken andre komponenter som henger sammen med den valgte komponenten, altså dens forgjenger og etterfølger. Dette kan løses på mange forskjellige måter, men den aller raskeste er at hvert komponent peker på sin forgjenger og etterfølger, altså en dobbellenket liste.

LinkedList og HashMap dekker og tilfredsstiller de krav som stilles til denne oppgaven, som for eksempel effektivitet og lønnsomhet. I denne oppgaven benyttes derfor disse to lineære datastrukturene hvor det er behov for mellomlagring. For mer detaljert beskrivelse av mellomlagring henvises det til kapittel 7.4.3.3 og 7.4.3.4.

Det brukes ingen listestruktur for å lagre elementene som ligger på tegnepanelet, fordi disse elementene legges inn i kontainerlisten som panelet allerede har. Mer om dette beskrives i underkapittelet 7.4.3.3.

ExtendsPoint inneholder kun 3 pekere per i dag, den ene pekeren peker til forrige mens de to andre er grener som kan gå fra denne, se figur 7.17. Dersom ExtendsPoint skal kunne ha flere enn 2 grener, kan det brukes en liste av type LinkedList for å holde rede på flere enn 2 pekere. Denne muligheten er tiltenkt som en oppgave for videre utvikling..

7.4.3.3 Lagring av elementene under kretskort tegning

Alle komponentene som legges på tegnepanelet vil lagres i en «Container»-liste som panelet allerede innehar. Windows, Panels, Frames og Dialog er de som arver fra Container[11]. Fra denne listen kan det alltid hentes og fjernes elementer. Dermed er det ikke nødvendig å gjøre denne jobben unntatt for InteractiveInput komponenter. Neste avsnittet vil forklare årsaken til dette.

Når brukeren starter simuleringen vil det alltid startes å traversere fra InteractivInput komponent. Derfor lages det pekere til disse komponentene

i en separat liste av typen LinkedList. Dette gjøres for hver gang brukeren legger InteractiveInput komponenter på panelet. Ved å gjøre dette unngås det søking av disse komponentene når en starter simulering. Dermed sparer vi tid, og antall sammenligninger vil reduseres i programmet.

7.4.3.4 Hjelpelister, Arrayer og HashMap

Digital Circuit inneholder også en liste av typen LinkedList for å lagre pekere til portene som velges samtidig. Denne listen kan brukes til å utføre en samtidigoperasjon «concurrent operation» for eksempel til flytting eller fjerning av disse elementene. Men denne funksjonaliteten er ikke implementert per i dag, og er tiltenkt som videreutviklingsarbeid. I dag brukes listen kun til å lagre en peker til den porten som kan flyttes.

Det er også opprettet en hjelpeliste av typen LinkedList, for å lagre peker til MovePoint komponentene som plasseres på panelet. MovePoint komponentene lagres i denne hjelpelisten til brukeren er ferdig med å utføre flytting. Når brukeren går ut av flyttemodus vil listen tømmes.

HashMap brukes som en hjelpeliste til å unngå at ledninger kan koble til seg selv. Ledningssegmenter har en unik nøkkel, og under oppretting av disse komponentene lagres nøklene sammen med ledningssegmentet i en HashMap. Når brukeren prøver å koble til et annet ledningssegment, vil nøkkelen til ledningssegment sammenlignes mot denne listen. Dette gjøres for å sjekke om ledninger ikke koples mot seg selv. Når det er utført en lovlig tilkobling fjernes elementene fra listen.

7.4.3.5 Dobbeltlenket liste

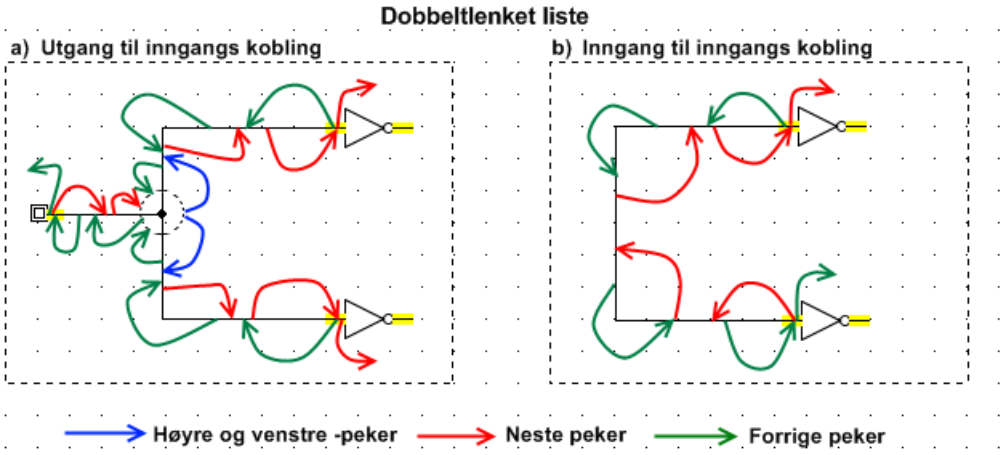
Komponentene er implementert med toveis pekere, forrige og en neste peker. Denne type liste kalles for en dobbeltlenket liste. Fordelen med å implementere en dobbeltlenket liste er at det alltid kan søkes i begge retninger fra et bestemt element. Plassering av en enkelt element i denne listen gjøres kun ved å oppdatere disse pekerne.

Når brukeren vil koble et ledningssegment til en annen ledning/ledningssegment, plasseres en ExtendsPoint-element i det punktet der ledningssegmentet koples til, og ledningen deles i to. For å plassere ExtendsPoint-elementet oppdateres kun pekere til ledningssegmentet brukeren trykker på. Dermed kan nye elementer plasseres i en bestemt posisjon med mindre antall sammenligninger og på kortere tid.

På figur 7.9 vises hvordan et dobbeltlenket pekersystem blir håndtert i Digital Circuit.

7.4.4 Implementering av ledningssegmenter i Digital Circuit

Det trengs to punkter for å bygge en ledningssegment. Første punkt forteller hvor i komponenten linjen skal startes, og det andre punktet forteller hvor



Figur 7.9: Dobbeltlenket liste

linjen skal avsluttes.

Når brukeren velger det første koblingspunktet i tegnepanelet, vil det trekke en linje fra det punktet og frem til musmarkøren. Så lenge brukeren ikke velger det andre koblingspunktet vil linjen henge med musmarkøren. Når brukeren angir det andre punktet, vil start- og sluttpunkt bli brukt til å opprette en ledningssegment.

7.4.4.1 WireComponent-klassen

WireComponent er komponenter av typen WireElement som implementerer grensesnittet «MouseListener» og «MouseMotionListener». Denne komponenten representerer ledningssegmenter i oppgaven.

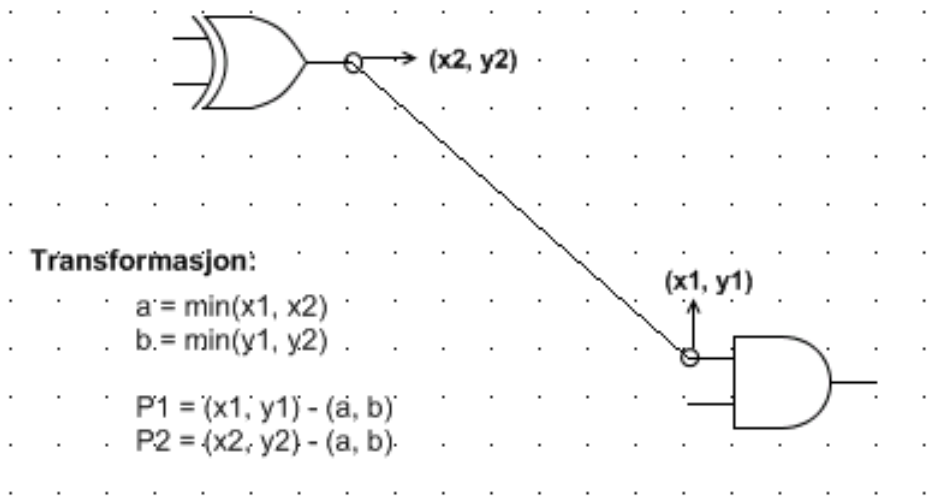
7.4.4.2 Transformasjon av koordinatsystem

De to punktene som blir sendt videre til å bygge en ledningssegment er i forhold til tegnepanelet som nevnt tidligere. Dermed må punktene transformeres i forhold til ledningssegmentet, se figur 7.10.

Figuren 7.10 viser hvordan utregning av punkter beregnes i forhold til ledningssegmentet. P1 og P2 på bildet er de nye punktene i forholdt til ledningssegmentet. For å finne dette punktet må en først finne minste x og y av de først kommende punkter. Deretter fås P1 og P2 ved å trekke fra minste x og minste y.

7.4.4.3 Implementering av WireComponent-klassen

Ledningssegmenter er implementert slik at det tegnes et rektangel rundt hver ledningslinje ved hjelp av GeneralPath, se figur 7.11. Innenfor dette



Figur 7.10: Transformasjon av punkter

området vil lyttere fungere, og det vil være enklere å treffe ledningslinjen med musmarkøren.

Dette rektangelet vil ikke være synlig for brukeren. Rektangelet brukes kun for å redefinere contains-metoden som avgjør det lovelige området til komponenten. Det gule området utgjør det uaktuelle området som brukeren ikke er interaktivt med, mens det grønne definerer det interaktive området som brukeren kan trykke på.

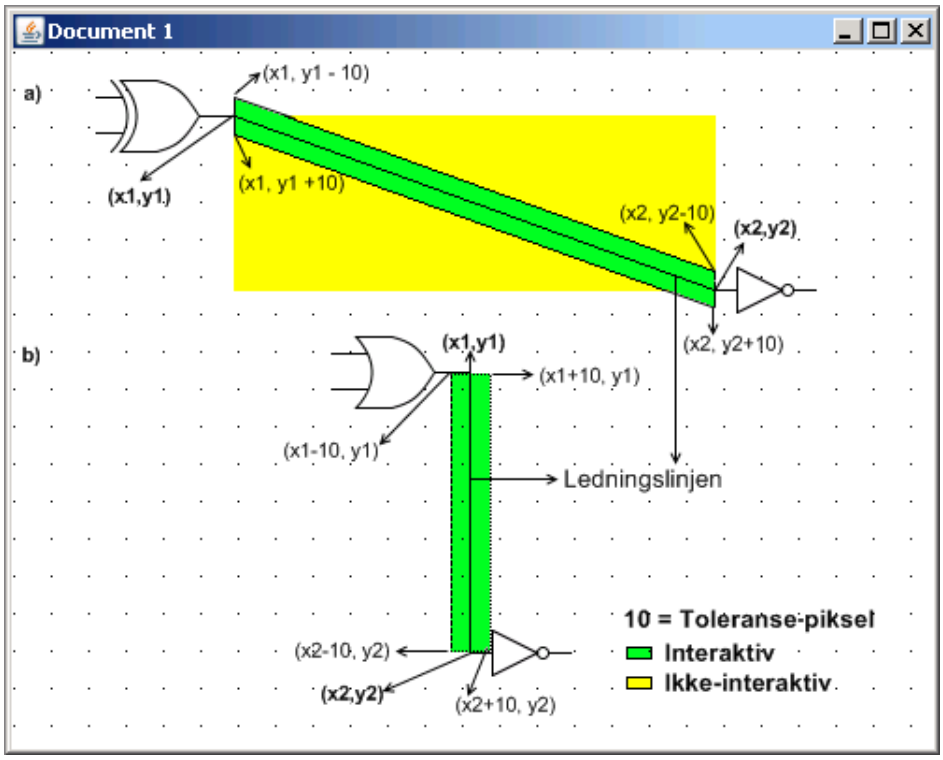
Illustrasjon 7.11 viser hvordan punktene ble bestemt for å tegne rektangelet rundt ledningslinjen. Situasjonene a og b på bildet viser hvilken hensyn som måtte tas i betraktning for å tegne riktig rektangelet. I situasjon a så er x1 og x2 ulik, dette medfører at vi må endre y for å få et riktig dekkende rektangel. Under situasjon b så er x1 og x2 lik og vi er nødt til å endre på x'ene for at rektangelet skal dekke hele ledningslinjen.

Det er bestemt i programmet at det skal være en toleranse på 10 piksel rundt ledningslinjen. Dette er området brukeren kan treffe med musmarkøren. Dersom det viser seg at dette ikke er nok kan denne utvides.

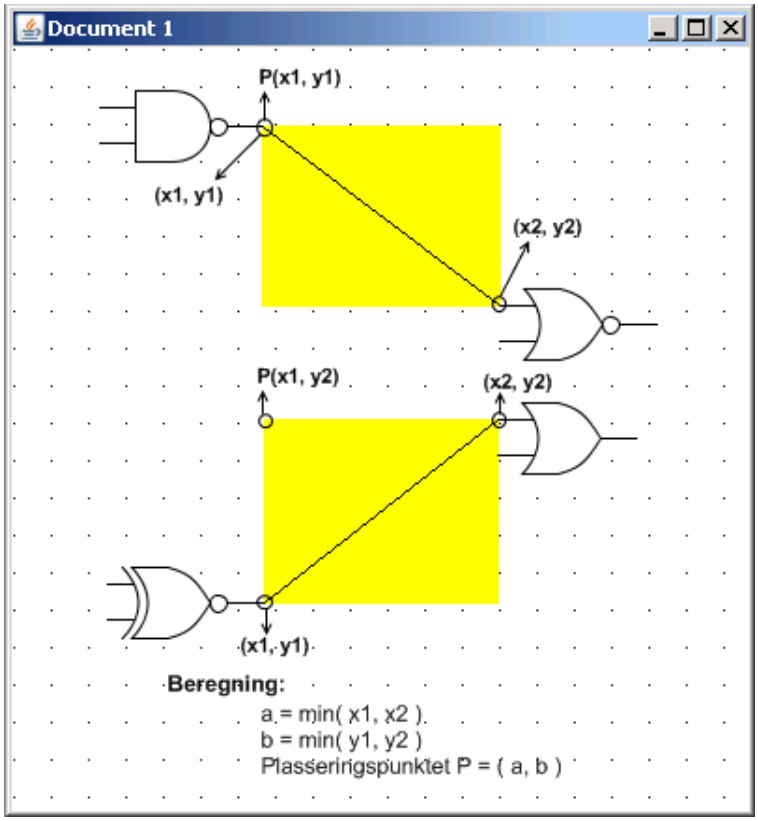
7.4.4.4 Plassering av ledningskomponenter

For å kunne plassere en komponent i Java trenges det et punkt som forteller hvor i panelet komponenten skal plasseres. Komponenten vil da plasseres fra dette punktet.

Når ledningssegmentet mottar punktene for å tegne ledningslinjen, vil komponenten samtidig beregne plasseringspunktet. Dette punket lagres i en egen variabel. Hvert enkelt ledningssegmentet vil inneholde dette punktet, som brukes under plassering/flytting av komponenten. Under flytting vil



Figur 7.11: Rektangelet rundt ledning



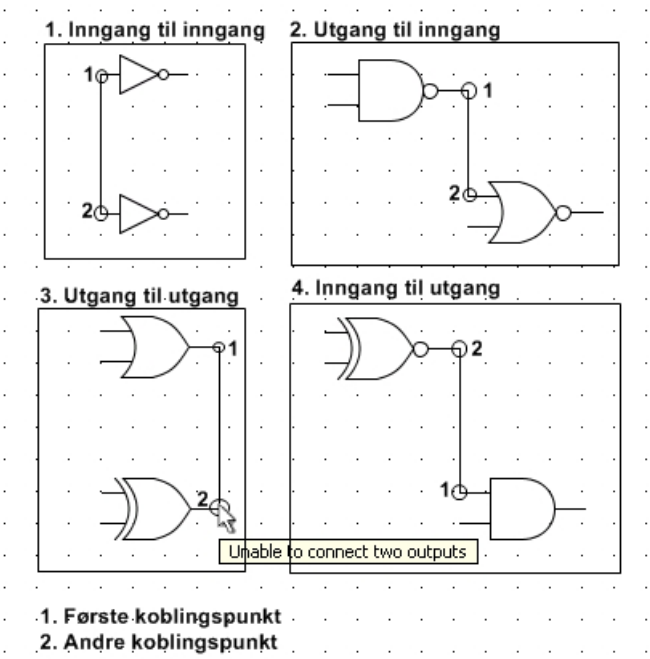
Figur 7.12: Plasserings punkter

plasseringspunktet oppdateres hele tiden slik at komponenten ligger på riktig sted til enhver tid.

I programmet er det implementert en metode for plassering av ledningssegmenter. Denne metoden krever kun en parameter av typen WireComponent (ledningssegment), og bruker plasseringspunktet som ligger i WireComponent-objektet.

Under beregning av plasseringspunktet brukes samme metodikk som i transformasjon av punkter. Derfor finner vi også dette punktet under beregning av transformasjonen, og dette punktet lagres i en egen variabel. Det vil si at all beregning utføres i en og samme metode. Se figur 7.12 for beregning av plasseringspunktet .

Punktet P er plasseringspunktet til ledningskomponenten på figur 7.12. Punktet P til en ledningskomponent fås ved å finne minste x og y, ut ifra mottatte punkter for å tegne linjen.



Figur 7.13: De fire reglene for kobling

7.4.4.5 De fire reglene for kobling

Input og Output ledningselementer er implementert med MouseListener grensesnittet, og er satt til å fungere etter bestemte regler. Figur 7.13 viser disse reglene. Legg spesielt merke til at den tredje koblingen på figuren er en ikke lovlig kobling. Kompling 1, 2, og 4 er de lovilige koblingene som tillates i Digital Circuit.

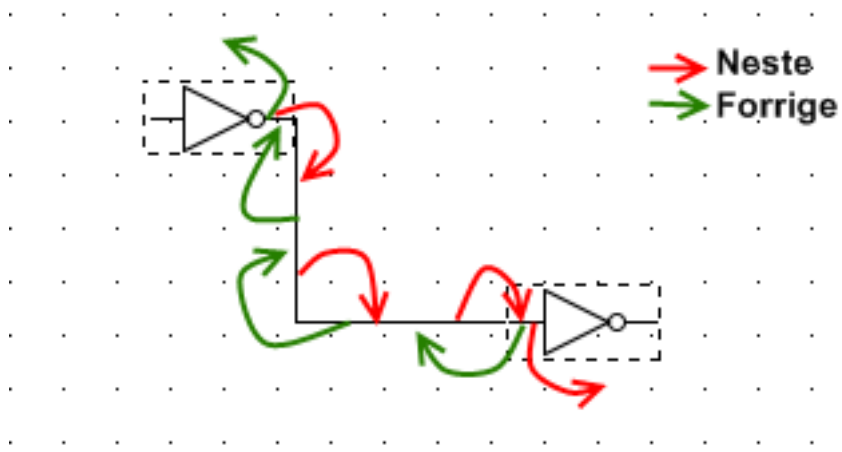
7.4.4.6 Kobling av Wire

Alle komponenter i Digital Circuit er bygget som selvstendige enheter. Ledningssegmenter, porter og andre elementer blir plassert slik at det ikke er noen mellomrom mellom disse. Det gjør at komponentene ser ut som om de er koplet sammen. For å kunne simulere dem, og for at de skal kunne henge sammen under flytting har det blitt benyttet en dobbeltlenket pekersystem.

7.4.4.7 Pekersystem

Superklassen WireElement er implementert med to pekere; en neste og en forrige peker. Etterkommere av WireElement vil da inneholde begge pekerne.

Pekersystemet er tiltenkt sånn at utgangsretningen peker framover og inngangsretning peker bakover. Dermed vil en utgang sin forrigepeker alltid



Figur 7.14: Pekersystemet mellom to porter

være null, men nestepeker er avhengig av om den er koblet til eller ikke.

En inngang vil normalt inneholde sin forrigepeker som peker til komponenten i retning bakover. Dersom nestepekeren til en inngang ikke er null betyr det at det er en inngang-til-inngang kobling. Dette vil si at det er en «inTOinType» -type kobling, se figur 7.15.

WireComponent er også av typen WireElement som har to pekere, forrige og neste. Et enkelt komponent av denne typen vil alltid peke til sin forrige og neste komponent, se figur 7.14.

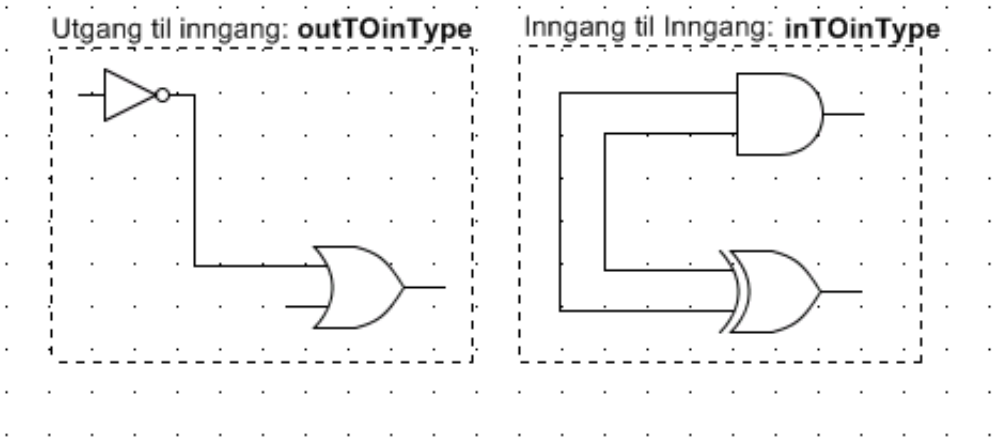
En ExtendsPoint inneholder tre pekere; forrige, høyre og venstre, som nevnt tidligere. Dette elementet brukes til å lodde delte kabler. Pekere muliggjør at elementene også henger sammen etter en lodding.

7.4.4.8 Flytting av ledningssegmenter ved hjelp av MovePoint

Flytting av ledningssegmentene i Digital Circuit skjer ved hjelp av et eget komponent som heter MovePoint. MovePoint komponenten er implementert for denne oppgaven og inneholder to pekere som er av typen WireComponent. Disse to pekerne peker til ledningssegmentene som dette MovePoint-komponent ligger over. MovePoint-komponenten inneholder to int typer som forteller noe om hvilket punkt som skal oppdateres under flytting. Siden en linje har to punkter er det kun ett av dem som skal oppdateres.

Disse MovePoint komponentene opprettes under flyttemodus og fjernes når brukeren går ut av modusen. Denne modusen gjenkjennes ved at brukeren trykker på ledninger. Når brukeren trykker utenfor disse ledningssegmentene går programmet ut av flyttemodusen.

MovePoint komponenten implementerer MouseListener og MouseMotionListener-grensesnittet, for å kunne lytte til brukeren. Når brukeren drar flyttepunktet begynner MovePoint komponenten å sende de nye punktene. Disse punkte-



Figur 7.15: To type koblinger

ne blir sendt til ledningssegmenter som MovePoint-komponenten peker til. Figur 7.16 viser tre MovePoint komponenter som er plassert over en kobling.

7.5 ExtendsPoint

En ExtendsPoint sin oppgave er å lodde ledningene som deles i to. Når brukeren er i kabelmodus og trykker på en ledningskomponent, vil den ledningen deles i to. ExtendsPoint komponenten blir plassert over de delte ledningssegmentene.

ExtendsPoint komponenten inneholder tre pekere; forrige, høyre og venstre, som nevnt tidligere. Disse pekerne bistår for flytting og simulering. Figur 7.17 viser en illustrasjon.

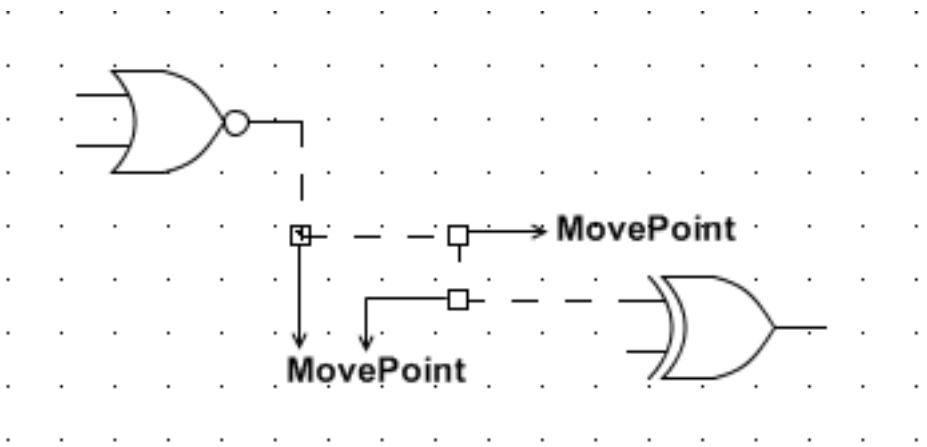
7.6 InteractiveInput

InteractiveInput komponenten er tegnet av GeneralPath, og satt sammen med en utgang. Denne komponenten har som funksjon å sende 1 eller 0, avhengig av det valget brukeren tar. I tillegg implementerer denne komponenten Runnable- og muslytter- grensesnitt.

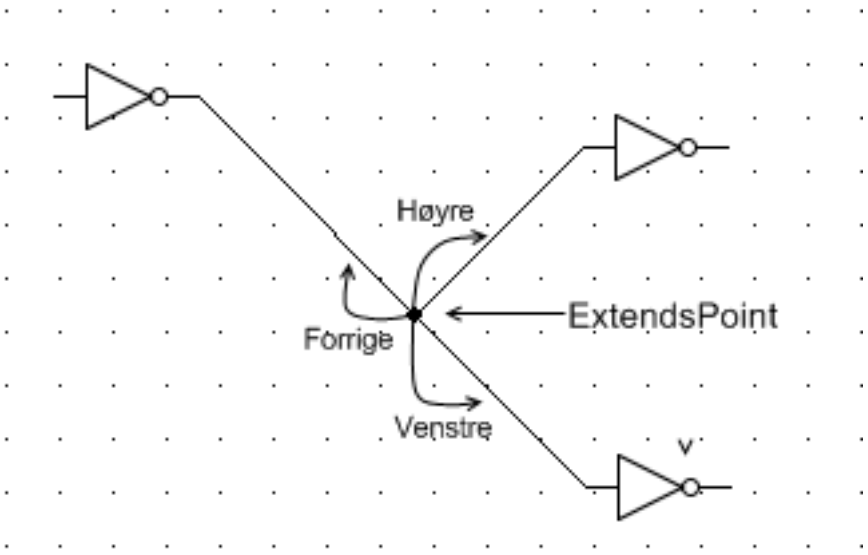
Dersom InteractiveInput komponenten er fylt med svart farge i midten, betyr det at signalet er 1 og hvis ikke er det 0. Figur 7.18 viser dette.

7.7 Led

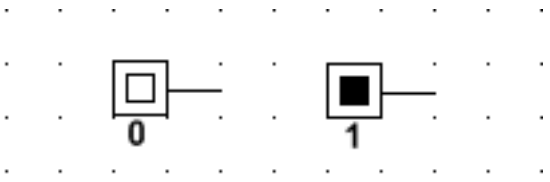
Led komponenten er tegnet av Ellipse2D og satt sammen med en Inngang. Denne komponenten har som funksjon å lyse når inngangssignalet er 1. Led-



Figur 7.16: MovePoint komponenter



Figur 7.17: ExtendsPoint komponent



Figur 7.18: InteractivInput komponenter

dioden vil kun lyse i simuleringsmodus, når «play» -knappen er trykket inn, dvs. at simuleringen er startet.

7.8 Makroer

En Makro er en komponent som bygges for å utføre en bestemt funksjon i en krets. Funksjonen implementeres som en vanlig krets ved hjelp av logiske porter og andre elementer. Deretter kan den implementerte kretsen pakkes inn som et eget makrokomponent.

En av de største fordelene med makroer er at gjentatte funksjoner i en krets, kan fås ved å bruke disse makroene. På denne måten vil det være mulig å bygge opp større og avanserte kretssystemer.

Digital Circuit oppretter makroer ved å åpne hvilke som helst tidligere lagret kretsfil til å gjenåpnes som en makro. For å kunne åpne kretsen som en makro, må kretsen bestå av InteractiveInput-objekter, og Led-objekter.

7.8.1 Oppbygging av Makro

I Digital Circuit er det implementert en klasse som heter «Macro» for å bygge opp makroer. Denne makroklassen tar imot panelet som inneholder kretsen og bygger opp en makrokomponent med innganger og utganger. For hver av de InteractiveInput-objekter som panelet inneholder vil det opprettes en inngang og for hver av de Led-objektene en utgang. Dette gjøres for å kunne sende inn og ut signaler til makroen.

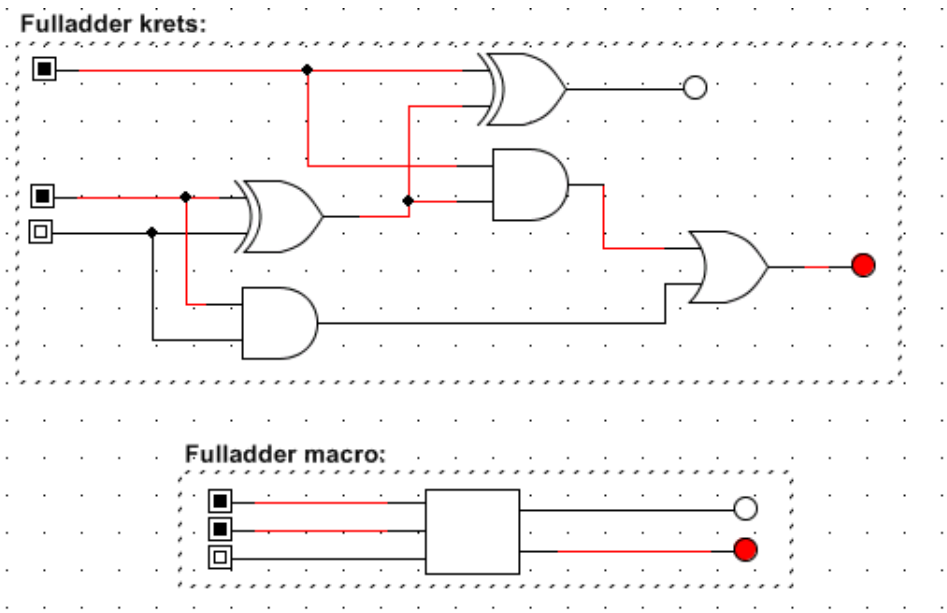
Makroen må ha like mange tilsvarende innsignaler som kretsen den inneholder, og like mange utganger. Om den mangler noen av disse, vil den ikke kunne utføre tiltenkt funksjonalitet.

Makroene tegnes som et rektangel ved hjelp av Rectangel2D, se figur 1. På høyre side av det rektanglet plasseres utgangene og på venstre side plasseres inngangene.

7.8.2 Oppdatering av pekere

Makroen er ansiktet utad for den kretsen den innehar. Signalene som makroen får på inngangene, må sendes videre til kretsen som makroen innehar. Når signalene som kommer til ledene i kretsen, vil de sendes videre til utgangene på makroen. Dette er det som må gjøres for at simuleringen av en makro kan fungere.

Dette blir gjort ved å oppdatere pekere i kretsen som makroen innehar, til å peke mot inngangene og utgangene på makroen. Etterkommere av InteractiveInput-komponenten blir koblet til inngangen som opprettes for denne makroen. Forgjengerne til Led-komponenten blir koblet mot utgangen som opprettes for dette.



Figur 7.19: Fulladder makro med Digital Circuit

7.8.3 Størrelsen av en makro

Størrelsen av en makro vil variere, avhengig av antall InteractivInput-komponenter og Led-komponenter en krets inneholder. Makro-klassen finner først antallet til hver av de to nevnte type komponenter, og deretter brukes det største tallet for å beregne størrelsen. Dette gjøres for å få plass til inngangene og utgangene som opprettes for InteractiveInput- og Led-komponenter. Størrelsen til en makro er definert da som; maks * 15 piksel.

På figur 7.19 vises det en fulladder-makro, som bygget opp av Digital Circuit.

7.8.4 Konklusjon

I Digital Circuit brukes makroer til å forenkle tegning av en bestemt krets, altså pakkes kretsen i en boks. Alle komponenter som er inn i denne boksen kalles for makrokomponenter, og som i helhet fremstiller en makro.

Programmet Digital Circuit har den funksjonaliteten som muliggjør hvilke som helst tidligere tegnet krets til å gjenåpnes som en makro. Denne løsningen vil bidra til at brukerne enklere forstår, og tar i bruk makroer.

Makroer fungerer ennå ikke helt optimalt i dette programmet. I noen tilfeller feiler kretsen ved kobling av makro til makro. Endringer som gjøres i filen som brukes til å opprette en makro, vil ikke føre til endringer i kretsen som bruker denne filen. I Digital Circuit er ikke makroer fullstendig testet, og alle mulige feilsituasjoner er derfor ikke avdekket. Dette er en del av det

viderearbeidet tiltenkt for programmet.

7.9 Lagring av kretskort

XMLEncoder eller ObjectOutput stream kan benyttes til å lagre kretstegningene i Digital Circuit. Her skal det drøftes om hvilken av dem som er mest hensiktigmessig til denne oppgaven.

7.9.1 Binærformat

Dersom det benyttes ObjectOutput stream vil filene lagres i binærformat. Denne type format er ikke leselig for et menneske. I tillegg vil filer som er lagret i binærtformat vil ikke være leselig for et program som har gjennomgått oppdateringer. Dette skjer fordi objekter som ble lagret i binærformat i en tidligere versjon av programmet, nå vil leses opp igjen i feil format grunnet oppdatering av de relaterte objektene.

7.9.2 XML -format

Når det benyttes XMLEncoder vil filene lagres i XML format. Denne type lagringsformat er leselig av et menneske, og tidligere lagrede filer kan åpnes i programmet som har gjennomgått oppdateringer. Dette er mulig fordi objektene blir opprettet på nytt når informasjonen lese fra XML-filen.

7.9.3 Konklusjon

Største fordelen ved å bruke XML lagringsformatet, er at lagrede filer er leselig for et menneske. Lagringsformatet vil kunne leses av programmet etter at det har vært gjennom oppdateringer. På grunn av disse fordelene velgers XML formatet til lagring. XMLEncoder brukes til å lagre fil i XMLformatet, ,mens XMLDecoder brukes til å dekode den lagrede filen. XMLEncoder og XMLDecoder ligger i Javas standardklasse biblioteket. Figuren 7.20 viser hvordan en lagret fil ser ut.

7.10 Konklusjon

Oppbyggingen av Digital Circuit er gjort med tanke på videreutvikling og med tanke på formålet vårt. I tillegg har skalering av porter, implementering av ledningssegmenter og valg av lagringsformat vist seg å være godt nok til vårt formål. Dette har jeg bekreftet ved å implementere alle disse ideene i Digital Circuit.

Etter min oppfatning kan pekersystemet utvides med flere pekere, på denne måten vil det være mulig å ha flere grener til en ExtendsPoint-komponent.

```
<?xml version="1.0" encoding="UTF-8"?>
<java version="1.6.0_11" class="java.beans.XMLDecoder">
 <array class="java.awt.Component" length="1">
  <void index="0">
   <object class="gates.NotGate">
    <void property="bounds">
     <object class="java.awt.Rectangle">
      <int>12</int>
      <int>0</int>
      <int>30</int>
      <int>24</int>
     </object>
    </void>
    <void method="add">
     <object id="Input0" class="gates.Input">
      <void property="bounds">
       <object class="java.awt.Rectangle">
        <int>0</int>
        <int>9</int>
        <int>13</int>
        <int>7</int>
       </object>
```

Figur 7.20: En lagret fil i Digital Circuit

Grunnet begrenset tid er koden ikke optimalisert, og dette kan videre forbedres.

Kapittel 8

Simulering

For at en student skal være sikker på at en krets virker slik den er tenkt, er det viktig å kunne simulere den. Simulering av digitale kretser har en stor grad av interaktivitet, det vil si at programmet reagerer raskt på de hendelser og endringer som brukeren gjør. For å få til simulering som best mulig etterligner en modell fra virkeligheten, bør programmet ha flere samkjørende og/eller ekte parallellkjørende sekvenser.

Algoritmene som leder frem til en løsning på de utfordringene jeg hadde i denne oppgaven, er utarbeidet av meg.

8.1 Multi-threaded applikasjon

Denne applikasjon er en multi-threaded applikasjon, se side 16. Når brukeren starter simuleringen, opprettes det en tråd for hver InteractiveInput-objekt som tegnepanelet inneholder. Trådene som opprettes vil avslutte på ExtendsPoint-elementet, på en port eller ved Led-elementet.

Trådene blir kjørt parallelt på maskiner med flere CPU-er, og applikasjonen vil her gå raskere. Programstrukturen/koden i Digital Circuit vil også bli bedre når applikasjonen er implementert som multi-threaded. Tråder utfører oppgavene selvstendige og stort sett uavhengige av hverandre, dette gjør i tillegg at programmet fungerer mer robust.

Normalt vil en krets inneholde flere InteractiveInput- og ExtendsPoint-elementer, det vil derfor være mest hensiktmessig å bruke flere tråder. På denne måten kan kretsen simuleres på en mer effektivt måte. I tilfeller der brukeren endrer inngangssignaler under simulering, vil en flertrådet løsning være mer fornuftig, og muliggjøre mer dynamikk i programmet.

Ulempen med en slik løsning er at debugging ikke er like enkelt å utføre som for en single-threaded applikasjon. Store og avanserte muti-threaded applikasjoner som denne trengs derfor å bli testet nøye. Denne applikasjonen har vært testet under ulike situasjoner, men en grundig test har ikke blitt utført grunnet begrenset tid.

Dersom denne applikasjonen hadde vært singel-threaded, ville tilbakekoblinger i en krets ha vært enklere å simulere. Men etter min oppfatning, med noe videreutvikling og med nøye testing, vil tilbakekoblinger i denne applikasjonen fungere mye bedre enn singel-threaded applikasjoner.

8.2 InteractiveInput oppretter tråder

InteractiveInput-elementet er implementert med Runnable-grensesnittet, som nevnt tidligere. Hvert InteractiveInput-element innholder sin egen tråd som brukes for å traversere fra dette elementet. Når brukeren legger til InteractiveInput-elementet på tegnepanelet, lagres det en peker til objektet samtidig i en egen LinkedList-liste. Dette gjøres for at det ikke skal være nødvendig å søke etter disse elementer ved simulering. Når det trykkes på play-knappen, starter tråden til alle InteractiveInput-elementene som ligger i listen.

En krets kan inneholde flere InteractiveInput-elementer. Fra hvert av disse elementene startes tråder som traverser gjennom kretstreet. Brukeren har også mulighet til å endre InteractiveInput-signalet dynamisk mens simuleringen pågår. Det er kun tråden til det InteractiveInput-elementet som brukeren endrer signalet på, som vil starte å traversere på nytt.

På figur 8.1 vises det et eksempel hvor brukeren har endret InteractiveInput-signalet for komponent a fra 0 til 1, og da starter kun tråden til denne komponenten. Denne tråden traverserer kretstreet og viser at led-dioden vil lyse. Dersom brukeren har endret signalet til InteractiveInput-komponent b fra 0 til 1, vil tråden til denne komponenten utføre jobben fram til ExtendsPoint-elementet. Fra ExtendsPoint-elementet vil det startes to nye tråder som tar seg av jobben videre.

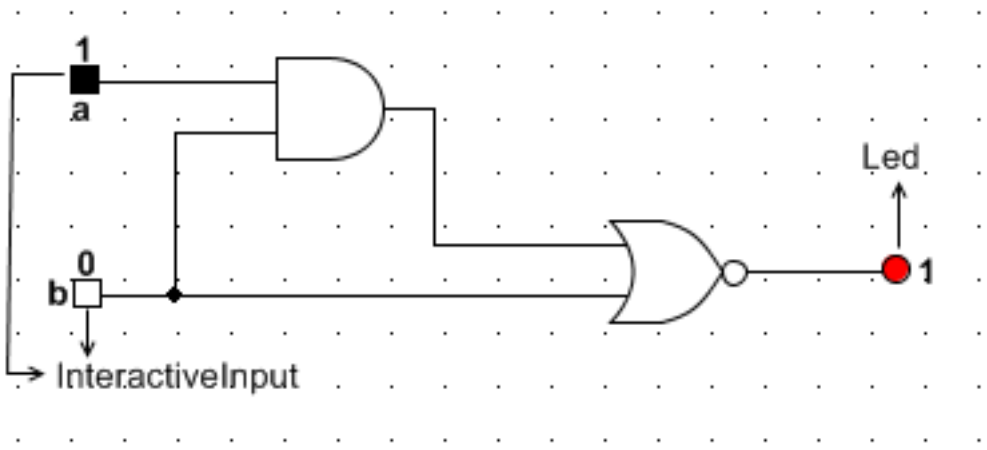
8.3 ExtendsPoint oppretter tråder

Fra et ExtendsPoint-element vil veien alltid dele seg i to retninger. Derfor er det logisk å opprette en tråd for hver retning. Ved å delegere oppgavene til to forskjellige trådene gjøres jobben parallelt og mer effektivitet.

Teknikken for å opprette tråder fra dette elementet kalles Adapter-teknikken, se kapittel 2.6.2.4 på side 24. Den synkroniserte startSimulering-metoden i ExtendsPoint implementerer denne teknikken.

8.4 Synkroniserte metoder

Når det opprettes flere tråder er det viktig å passe på at felles ressurser ikke aksesseres av mer enn en tråd om gangen. Derfor er det implementert synkroniserte metoder for å unngå «race condition».



Figur 8.1: Simulering

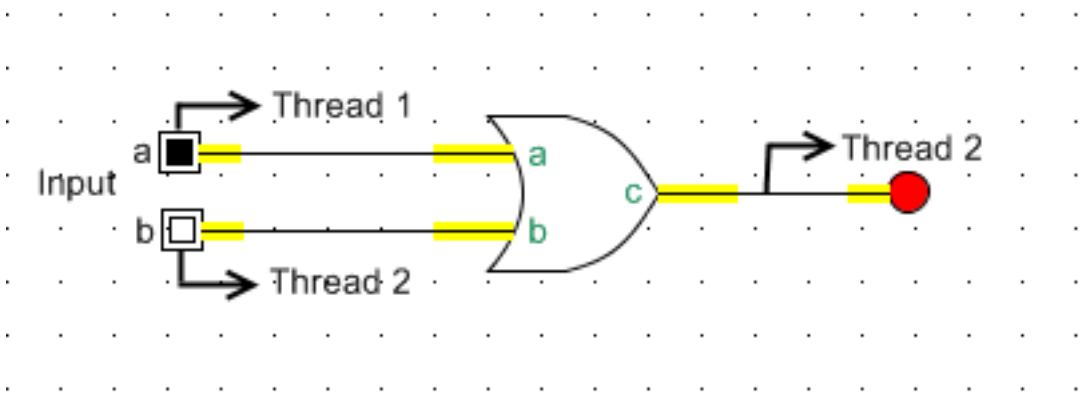
Hver port i programmet er implementer med startSimulering-metode som er synkronisert. Det er denne metoden som regner ut utgangsignalet. Utgangsignalet er avhengig av funksjonaliteten og inngangsignalet til porten. Hver gang det kommer et signal til inngangen, startes startSimulering-metoden for porten. Siden en port kan inneholde flere innganger kan det hende at to tråder kommer inn samtidig og endrer felles ressurser. Dermed er det nødvendig å synkronisere denne metoden.

ExtendsPoint-kabelelementet er også implementert med synkronisert startSimulerings-metode, slik det er nevnt tidligere. Hver gang vi kommer til et slikt element, vet vi at fra dette elementet blir veien delt i to. Da må vi traversere gjennom både høyre og venstre gren. For at vi ikke skal ødelegge pekere under traverseringen, har vi implementert denne metoden til å være synkronisert.

8.5 sendSignal-metoden

Hver port er også implementet med en sendSignal-metode. Når startSimulering-metoden har regnet ut utgangsignalet, kalles sendSignal-metoden inne i startSimulering-metoden. Denne metoden tar seg av traverseringen videre med utgangsignalet. Under traverseringen registreres signalet i de kabelelementene den kommer over.

Denne metoden er implementert av alle portene i Digital Circuit.



Figur 8.2: To tråder

8.6 Trådlogikk

I dette kapittelet skal vi se på trådlogikken i Digital Circuit. Det å opprette tråder i Java er ikke en vanskelig oppgave. Det viktigste og mest krevende er å få trådene til å samarbeide under simuleringen, slik at simuleringen gjøres på en mer riktig og enklere måte.

Hovedtanken bak trådlogikken min er at antall instruksjoner som utføres av prosessoren skal reduseres.

8.6.1 Hovedideen

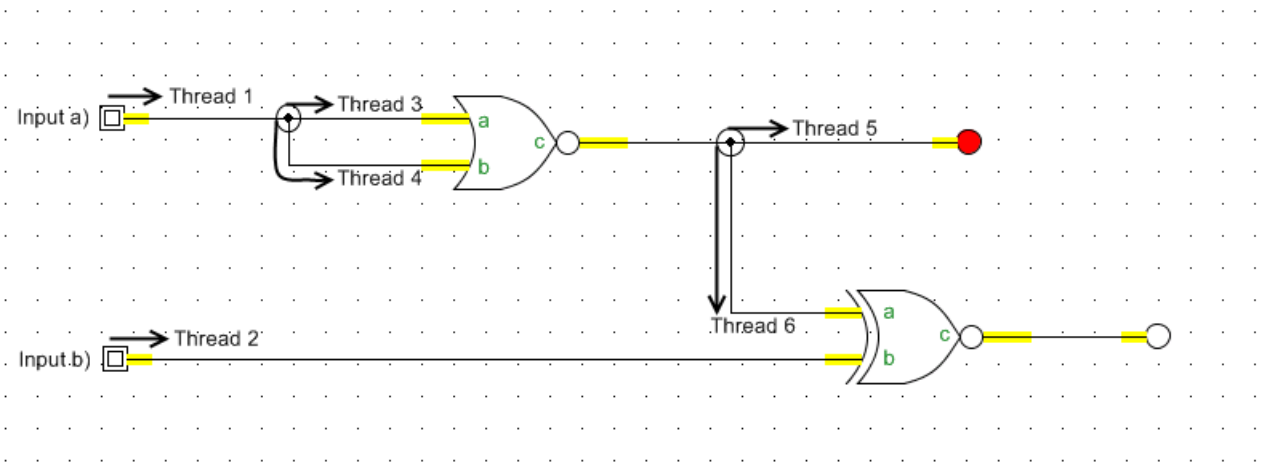
I figur 8.2 vises det en krets hvor signalet til OR-porten kommer fra to forskjellige InteractivInput-elementer.

Hvis det kobles to InteractiveInput-elementer til en port slik figur 8.2 viser, vil de to trådene som startes fra InteractiveInput gå gjennom utgangen til inngangene til porten. Problemet er at porten simuleres to ganger og dermed går utsignalet også to ganger fra porten. Selv om dette ikke påvirker resultatet vil hovedtanken bak logikken ikke oppfylles.[1]

Dette problemet er løst i Digital Circuit ved at programmet lar tråd 1 komme, registrere signalet og deretter dø. Tråd 2 kommer etter og registrerer signalet slik at simuleringen fortsettes.

Logikken her er at selv om det kommer to tråder til inngangene for porten, må det ikke tillates å simulere porten to ganger. Det vil være unødvendig tidsbruk dersom dette skjer. Derfor vil trådene som kommer inn, først sjekke om de andre inngangene er koblet til, og hvis den er koblet til, om det er registrert noe signal. Hvis ikke signalet er registrert, vil trådene dø og den siste tråden vil simulere porten og forsette simuleringen.

[1] Hvis det ikke legges begrensinger gir man blafring på skjermen, men det endelige resultatet blir ok.



Figur 8.3: Flere tråder under simulering

8.6.2 Større krets med flere tråder

Så langt har vi sett på en enkelt port med to tråder. I denne delen av kapittelet skal vi se på en større krets og de problemer som kan oppstå under simuleringen.

På figur 8.3 vises det en større krets som er tegnet av Digital Circuit og illustrerer de trådene som opprettes under simuleringen. Logikken her skal fungere på samme måte som det tidligere eksempelet.

For å forstå problemet som kan oppstå, definerers det to situasjoner:

- Når play-knappen trykkes
- Endring under simulering

8.6.3 Situasjon 1

Når brukeren trykker på play-knappen vil i utgangspunktet alt være nullstilt. Dermed vil det være nok å sjekke om inngangene er koblet til og om det er registrert signal. Altså det vi gjorde i kapittel 8.6.1.

8.6.4 Situasjon 2

Problemet kan igjen forekomme når brukeren endrer signalet under simuleringen. Når brukeren trykker på play-knappen har alle førstegangstråder registrert signalene i porten. Logikken for å sjekke om signalet til porten er registrert, slik det ble beskrevet i kapittel 8.6.1, vil derfor ikke fungere. På grunn av dette vil flere tråder i denne situasjonen simulere porten og forsette videre.

8.6.4.1 Utfordringer under situasjon 2

Problemet som oppstår i denne situasjon er at flere tråder kan gå gjennom utgangen til en port, selv om signalet opprinnelig kommer fra en og samme foreldre. På figur 8.3 vises en kretstegning hvor signalet til NOR-porten kommer fra InteractivInput a, det vil si fra samme foreldre. Det har blitt bestemt av meg at en ExtendsPoint-elementet ikke skal kunne fremstilles som en foreldre i kretsen. Jeg har valgt å legge inn denne logikken fordi jeg hovedsakelig er interessert i hvilken port eller InteractiveInput et signal har sitt utspring fra.

Både tråd 3 og 4 kommer inn i NOR-porten. Selv om begge trådene kommer inn skal porten kun simuleres én gang, og kun én tråd skal få lov til å forsette videre. Brudd på dette medfører at hovedtanken bak min trådlogikk ikke oppfylles.

8.6.4.2 Foreldre-algoritmen

Dette problemet løses ved å sjekke hvilke av inngangene til porten som har samme foreldre, i tillegg til å sjekke om inngangsignalet er registrert. Se figur 8.3 hvor NOR-porten har interaktiv input a som foreldre til begge inngangene. Av de inngangene som har samme foreldre må det ha vært registrert like mange antall signaler. Hvis ikke betyr det at en annen levende tråd finnes i programmet.

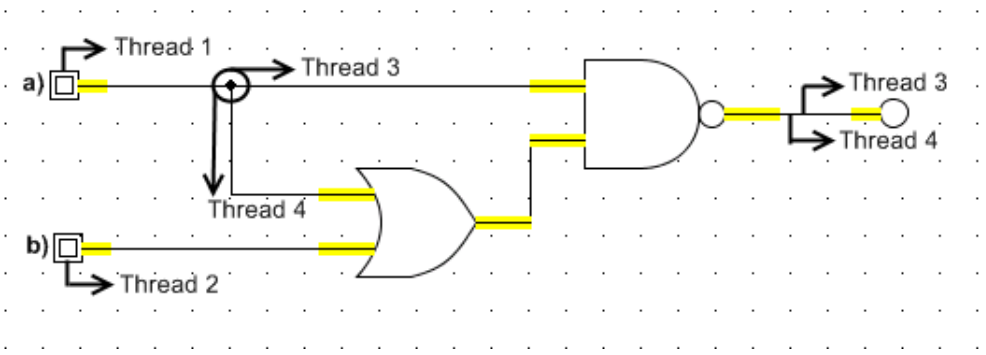
På figur 8.3 vises det et eksempel hvor simuleringen skjer med riktig logikk, slik det forventes. Fra NOR-porten forsetter tråd 4 gjennom utgang c og fra XNOR-porten fortsetter tråd 6 gjennom utgang c. Resten av trådene forsvinner underveis utenom tråd 5, som også viser inngangsignalet til led-dioden den er koblet mot.

Hvert element i programmet har en unik id (identifikator). Under simulering brukes denne identifikatoren i et Input-element for en port, til å markere hvem som er foreldre til inngangen. Ved å bruke disse identifikatorene kan denne algoritmen realiseres.

8.6.4.3 Problemet med «Foreldre-algoritmen»

På figur 8.4 vises det en situasjon hvor inngangene til NAND-porten har forskjellige foreldre, men samme rotkilder. Dette er et spesielt tilfelle hvor brukeren kan endre signalet til InteractiveInput a under simulering. Dersom dette skjer vil to tråder føres ut av NAND-porten. Den første inngangen til NAND-porten har InteractiveInput a som foreldre, mens den andre inngangen har en OR-port som foreldre. I utgangspunktet har inngangene til NAND-porten en felles rotkilde. Merk her at den ene inngangen til NAND-porten i tillegg har en annen rotkilde.

I henhold til «Foreldre-algoritmen» sjekkes det om inngangene har felles foreldre, og om det er registrert like mange antall signaler fra disse.



Figur 8.4: NAND-porten har samme rotkilde

Dersom dette ikke oppfylles, forsetter ikke tråden. Men her eksisterer det et spesialtilfelle som må tas hensyn til. Når inngangene til portene har samme rotkilde men ulike foreldre, må det sjekkes om det er registrert like mange signaler.

«Foreldre-algoritmen» kan utvides til en «Rot-algoritme» for å takle denne situasjon.

8.6.4.4 Rot-algoritmen

I logikken bak «Rot-algoritmen» vil det til enhver tid, under en simulering, være slik at elementer i en krets vil ha oversikt over tilhørende rotkilder. Rotkilden i Digital Circuit vil alltid være InteractiveInput-elementer, siden det er kun disse som kan sende ut signaler. Det finnes to viktige tilfeller som må ta hensyn til:

Tilfelle 1: Når inngangen til en port kun har én rotkilde. I en slik tilfelle vil det kun komme signal fra den ene rotkilden. På figur 8.5 vises det en AND-port som har kun én rotkilde til hver av sine innganger.

Tilfelle 2: Når en inngang har flere rotkilder. I en slik situasjon kan en inngang få signal fra en hvilken som helst, men kun fra én i et gitt øyeblikk, av rotkildene[2]. For eksempel se figur 8.5, hvor den ene inngangen til XOR-porten både har a og b som rotkilde.

8.6.4.5 Rot-algoritme som en løsning

For å implementere løsningen kan vi først sette opp kravene:

- Hver inngang til en port må kunne lagre flere id (identifikatorer) av rotkilder.

[2]En inngang koblet direkte til en InteractiveInput, altså en rotkilde, vil kun ha denne som sin rotkilde og ingen andre.

- Det må også registreres antall innkommende signaler fra en bestemt rotkilde.

- I kretsen må det til enhver tid, under simulering, opplyses om hvor endringene ble gjort.

En HashMap kan benyttes til å ta vare på flere id-er av rotkildene, sammen med tilhørende antall registrerte signaler, i et inngangselement. På grunn av at id-ene er unike, vil dette være en av de beste datastrukturer å bruke i denne sammenheng. I tillegg må traverseringsmetoder registrere id-ene til rotkilder i de elementene den kommer over under simulering. De enkelte inngangselementer vil da inneholde oversikt over de rotkildene den tilhører. En annen viktig oppgave å gjøre under simulering er å øke antall registrerte signaler i listen for den bestemte rotkilden.

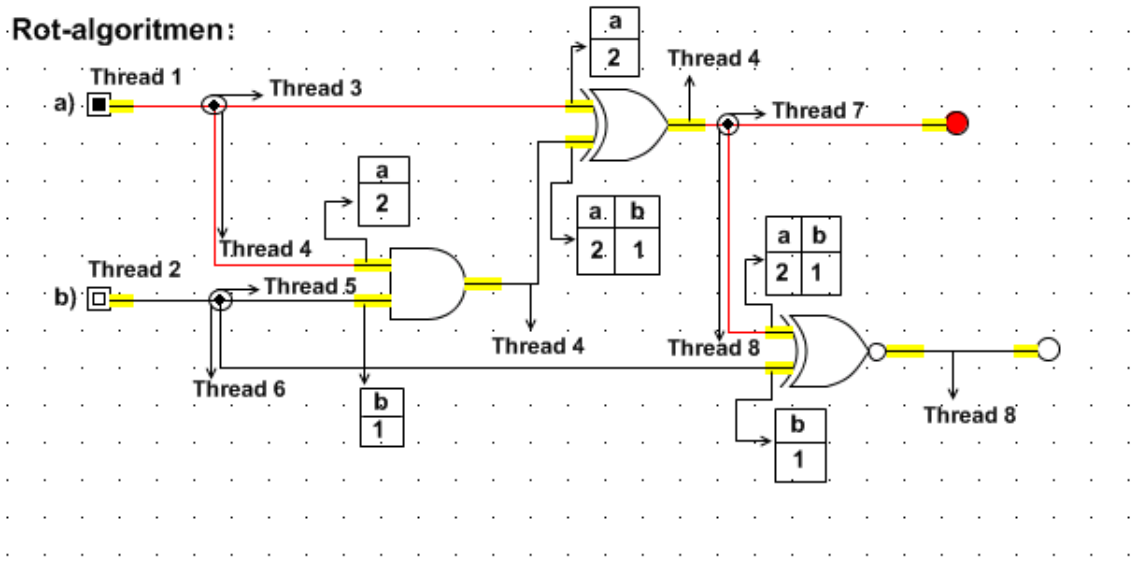
Når brukeren har trykket på play-knappen, ville inngangene til portene ha en liste over rotkildene de tilhører, som figur 8.5 viser. Når en endring påføres i kretsen ved å endre inngangssignalet til InteractiveInput-elementet, vil tråden til elementet startes for å traversere igjen, som nevnt tidligere. Denne tråden som startes igjen, vil ta med seg rotkildeid-en gjennom kabel-elementene. Når den kommer til inngangen, vil den øke antall registrerte signal med én for den innkommende rotkilden.

Etter å ha øket med én for antall registrerte signal, vil tråden sjekke de andre inngangene til den bestemte porten med den innkommende rotkildid-en, for antall registrerte signal. Hvis antall registrerte signaler avviker for de andre inngangene med den samme rotkildid-en, vil den aktuelle tråden avsluttes. Med dette kan vi konkludere at en eller flere av inngangene har samme rotkilde. Tråden vil da avsluttes fordi noen av de andre inngangene fortsatt venter på signal fra samme rotkilde.

Den tråden som fortsetter fra en bestemt port vil innholde unionen av listene til inngangene. På denne måten vil den ta med seg en liste over rotkilder som alle etterfølgende elementer må registreres med. Det vil si at etterfølger av den bestemte porten også vil ha samme rotkilder som forgjengere. På figur 8.5 vises det at en XOR-port sin andre inngang har fått både a og b som rotkilder.

8.6.4.6 Argumentasjonene for «Rot-algoritmen»

Det opprettes en tråd fra hvert InteractiveInput-element. Denne tråden kan enten fortsette direkte til inngangen for en port eller så kan to nye tråder opprettes i et ExtendsPoint-element. Da vil tråden fra InteractiveInput-elementet dø etter at den har opprettet to nye tråder til å forsette simuleringen, se figur 8.6. Trådene som opprettes av ExtendsPoint-elementet er for grenene som kan gå fra denne. De to trådene kan enten videre gå til et nytt ExtendsPoint-element, eller til en annen inngang. Merk her at begge



Figur 8.5: Rot-algoritmen

trådene samtidig ikke kan gå videre til en og samme inngang, fordi hver tråd vil gå til sin egen gren.

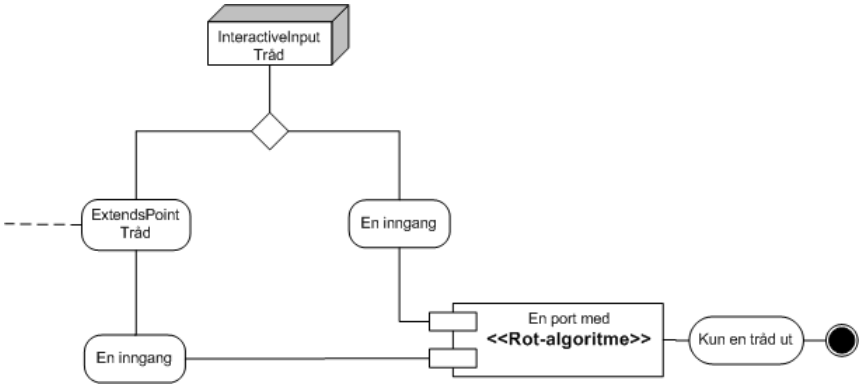
Når en port har flere innganger, kan det tenkes at flere tråder samtidig kan komme til en port. Dette stemmer! Det som igjen er viktig å huske er at, som nevnt tidligere, selv om det kommer flere tråder til den bestemte porten er det kun én av dem som samtidig kan utføre operasjoner, på grunn av at metodene for simulering er implementert til å være synkronisert.

Hvis trådene som kommer til en port fortsetter videre gjennom utgangen, blir det brudd på trådlogikken. «Rot-algoritem» sjekker hvem av de trådene som kommer inn kan forsette. Det vil si algoritmen sjekker om det er noen andre innganger i én og samme port, som venter på signal fra samme rotkilde. Når dette stemmer, vil den aktuelle tråden avsluttes. Dermed kan jeg konkludere at «Rot-algoritmen» alltid vil fungere i de ulike situasjoner.

Det er kun én tråd som kan komme til inngangen i en port gjennom én simulering. Når det kan komme en tråd til en inngang, vil det si at det kan komme flere tråder til en port, siden en port kan innholde flere innganger. Ved å sjekke om det er noen andre innganger som venter på signal fra den samme rotkilden, unngås det å simulere porten flere ganger og det er kun én tråd som alltid vil forsette fra en port ved en simulering.

8.7 STOP-knappen

Når brukeren trykker på stop-knappen, vil alle de aktuelle variablene nullstilles, slik at simuleringen skal fungere på samme måte ved neste



Figur 8.6: Forløp av tråder i en Rot-algoritme

simulering. De viktigste endringene programmet gjør er at inngangssignaler forandres til å ikke være registrert, led og ledningselement settes til å ikke lyse.

8.8 Alternative løsning

En alternativ løsning til å simulere en krets ville ha vært å bruke en enkel tråd med en graf-algoritme. Denne type løsning vil fungere slik at en og en gren i kretstreet må traverseres. Dersom tråden kommer til en ExtendsPoint, kan f.eks. den høyre grenen legges i en kø og etter at den venstre grenen er ferdig, kan den som ligger i køen tas ut og traverseres.

Singel-thread løsning vil fungere tregere på flere CPU-maskiner enn Multi-threaded, som nevnt tidligere. I tillegg vil denne løsningen føre til at det blir mindre dynamikk ved endring av signaler i en krets.

8.9 Konklusjon

«Rot-algoritmen» fungerer og oppfyller trådlogikken slik den er definert av meg. Ved å benytte denne algoritmen utføres simuleringen på en riktig og mer effektiv måte. Denne algoritmen er implementert i Digital Circuit og testene viser at algoritmen i programmet fungerer i alle tilfellene.

Del III

Avslutning

Kapittel 9

Oppsummering og videreutvikling

Digital Circuit oppfyller de kravene vi stilte til denne applikasjonen. Etter min mening er applikasjonen brukervennlig, og har også implementert de mest vesentlige funksjonene som trengs av en slik type applikasjon. Under oppbygging av applikasjonen ble det tatt hensyn til mulighetene for videreutvikling, og det er enkelt å implementere nye funksjoner som angre slik at man kan gå et steg tilbake, sletting av porter og «snap to grid».

Det er alltid behov for videreutvikling av denne type applikasjoner i takt med teknologisk utvikling og krav fra undervisningsmiljøer. Kildekoden vil gjøres tilgjengelig for «Open Source»-miljøet ved å publiseres på et «open source»-nettsted. Kildekoden skal legges ut med GNU-lisens.

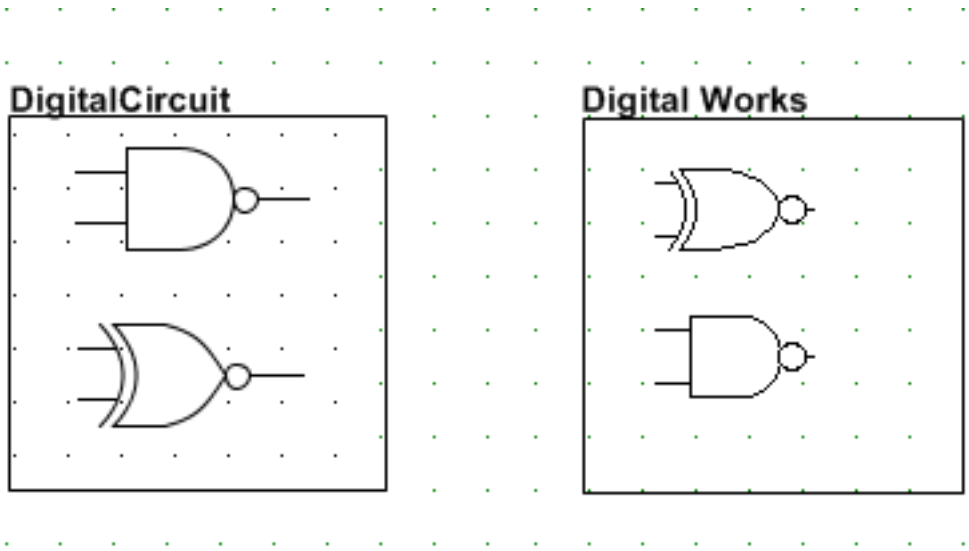
9.1 Portene

Etter min oppfatning har jeg ved å benytte Java oppnådd et penere bruker-grensesnitt enn Digital Works. Portene i Digital Circuit blir glattere og penere enn Digital Works ved å bruke RenderingHints sammen med Graphics2D sine innebygde funksjoner, slik vi kan se i figur 9.1.

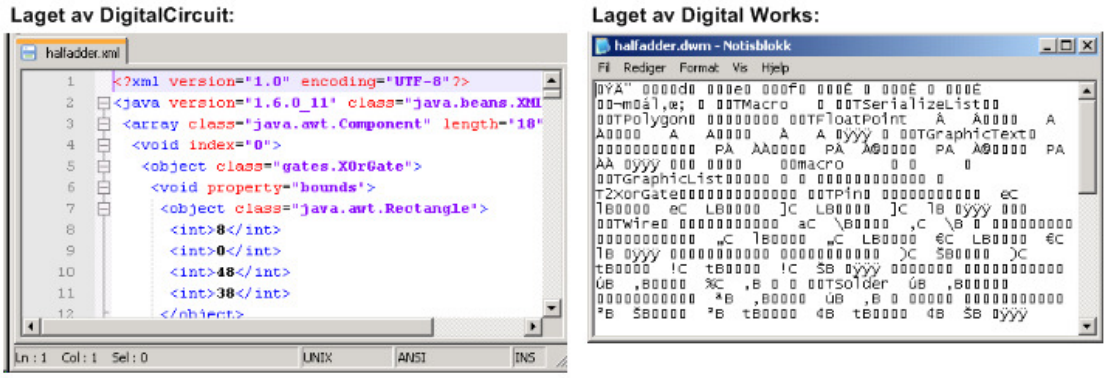
9.2 Lagring av kretser

Digital Circuit bruker XML som lagringsformat og Digital Works bruker en slags «ObjectOutputStream» til å lagre kretstegningene. Lagrede filer fra Digital Circuit er derfor leselig av mennesker, og men ikke de til Digital Works.

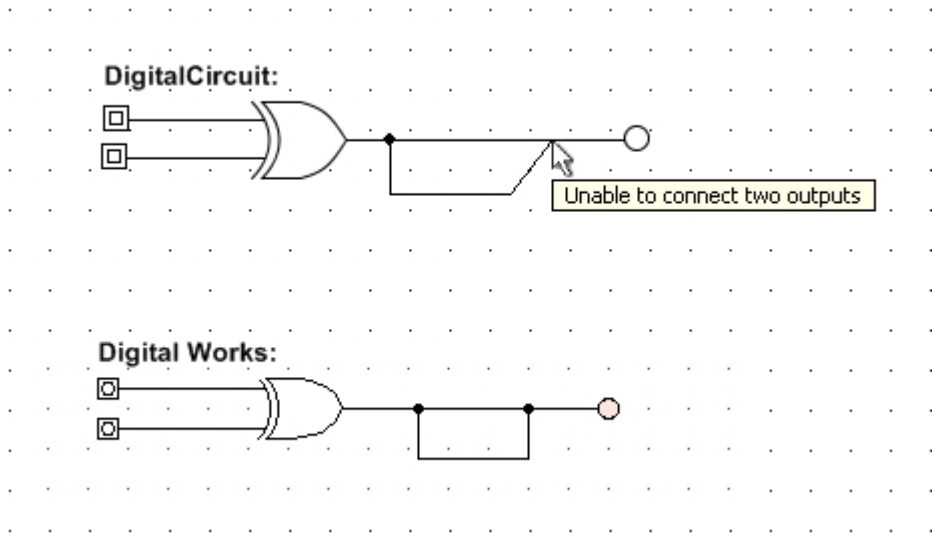
Filen lagret av Digital Circuit er også mulig å åpne i hvilket som helst operativsystem og robust mot oppdateringer av programmet. Figur 9.2 viser en sammenligning av lagrede filer fra med Digital Circuit og Digital Works.



Figur 9.1: Sammenligning av porter



Figur 9.2: Sammenligning av lagringsformat



Figur 9.3: Utgang-til-utgang kobling

9.3 Utgang til utgangskobling

Utgang til utgang er ikke en lovlig kobling i digitale kretser og derfor tillates ikke dette i Digital Circuit. Digital Works tillater ikke dette heller, unntatt det som vist på figur 9.3. Jeg antar at dette er en liten feil i Digital Works som gjør det mulig å koble fra utgang til utgang.

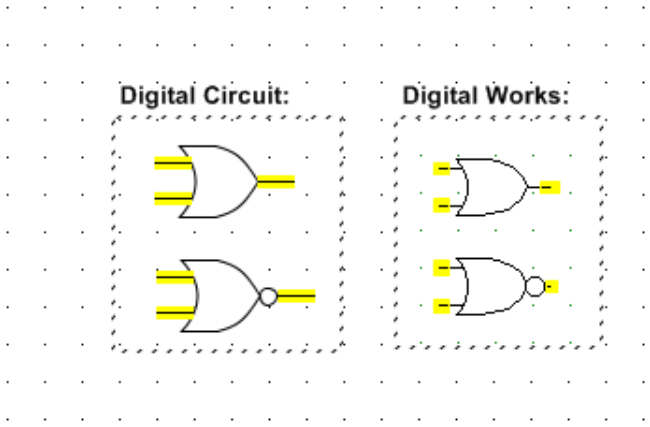
9.4 Akseptabelt område for tilkoblinger

Figuren 9.4 viser at akseptabelt område for tilkoblinger i Digital Circuit er større enn Digital Works. Det gule området vist for Digital Works er angitt i henhold til mine erfaringer.

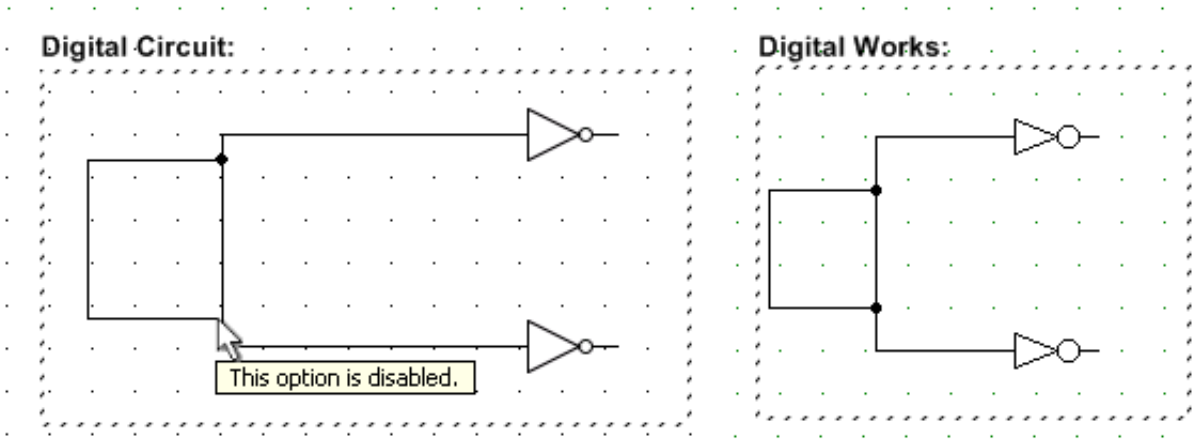
Portene i Digital Works er av samme størrelse unntatt NOT-port, dermed blir en XNOR-port sitt utgangskoblings-område blir enda mindre enn XOR-porten. Det gjør at kobling av ledninger blir vanskeligere i noen tilfeller som dette. I Digital Circuit er ikke akseptabelt område avhengig av type port. Det gule området på figuren 9.4 viser akseptabelt område for tilkoblinger.

9.5 Inngang til inngangskobling

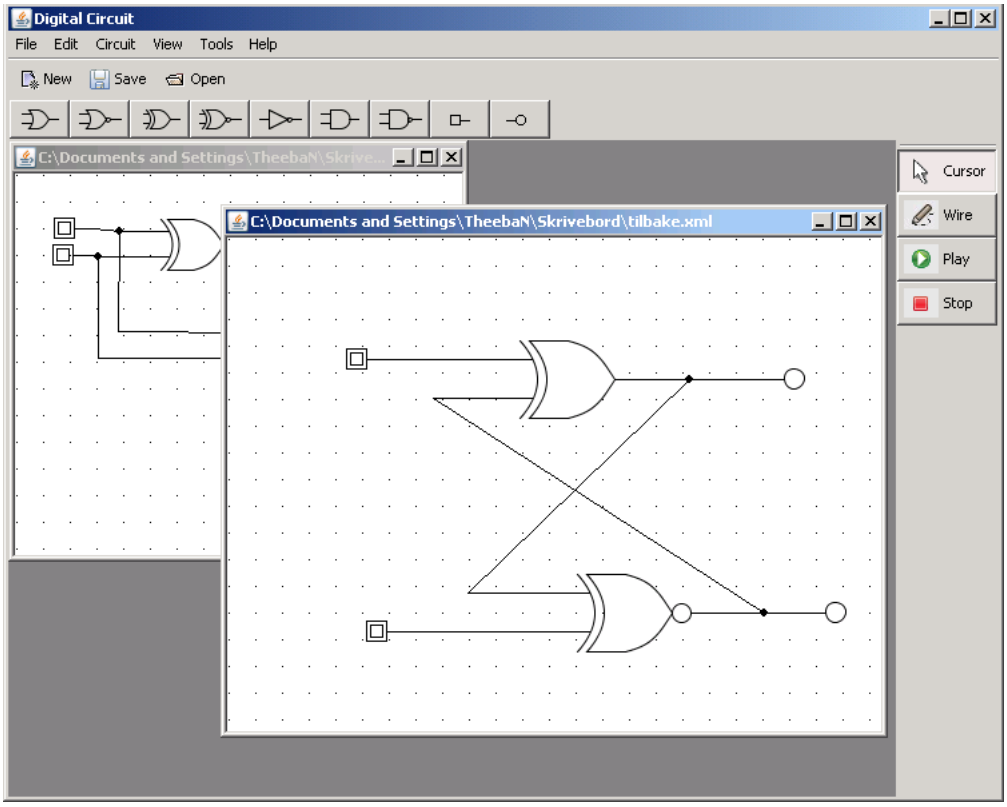
Input til input er en lovlig kobling i digitale kretser. I Digital Circuit gis det mulighet til dette for kun én engang. Dette vil det si at brukeren har ikke mulighet til å koble fra en inngang-til-inngang kobling til seg selv igjen. Dersom brukeren prøver dette, får en feilmelding som figur 9.5 viser. Digital Works tar ikke hensyn til dette, se figur 9.5.



Figur 9.4: Akseptabelt område for tilkoblinger



Figur 9.5: Inngang-til-inngang kobling



Figur 9.6: Flere krets samtidig

9.6 Flere kretser samtidig

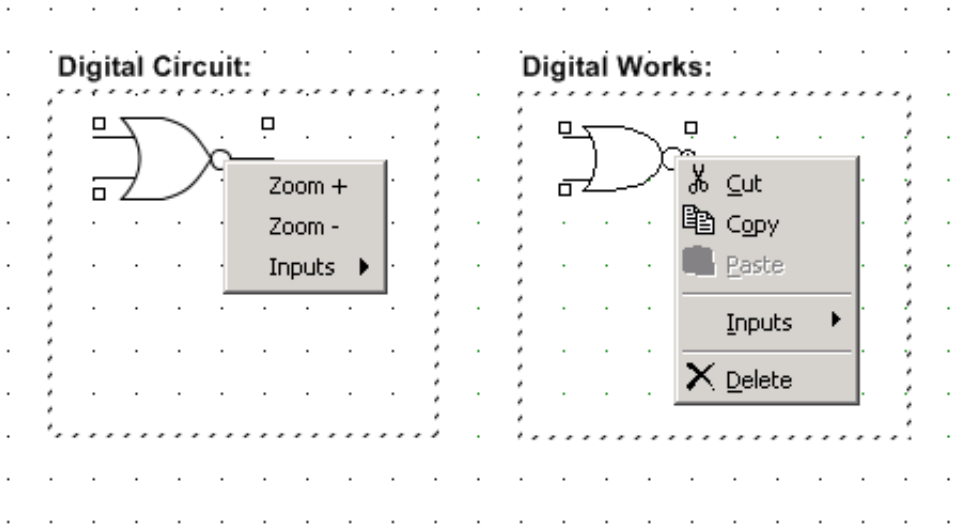
Digital Circuit gir brukerne mulighet til å jobbe med flere kretstegninger samtidig, se figur 9.6, men i Digital Works er ikke dette mulig.

9.7 Zoom in/out-mulighet

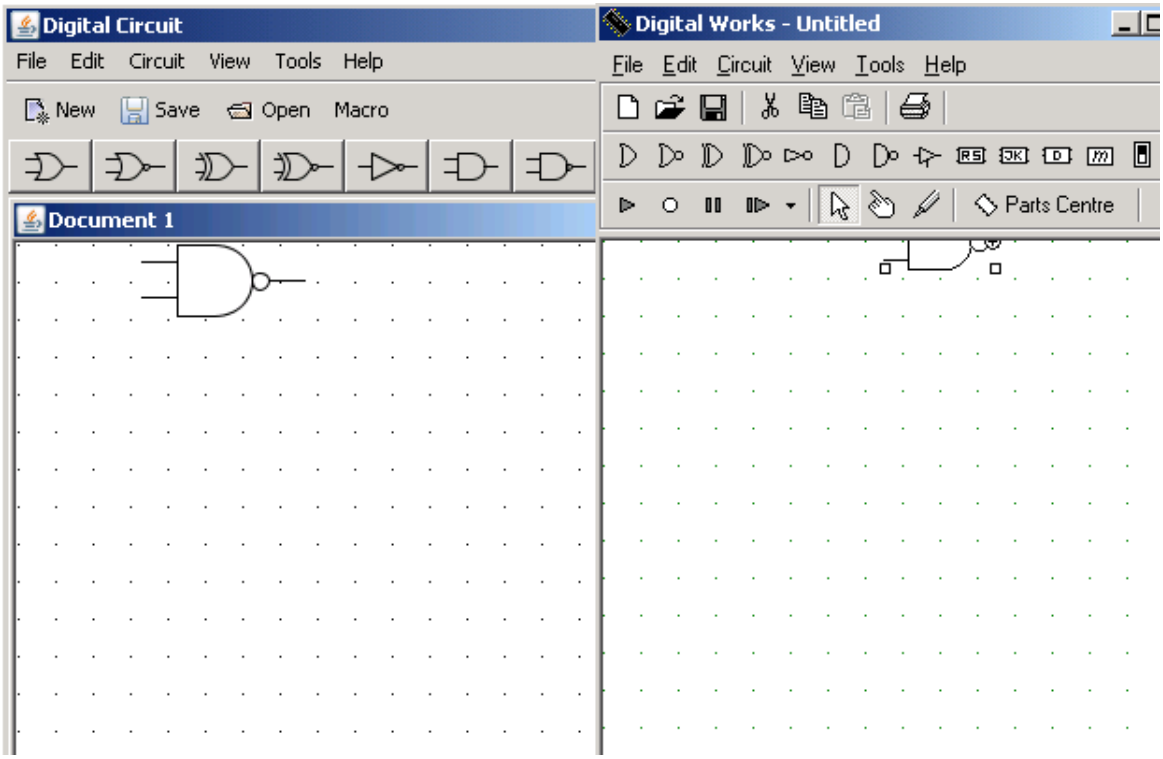
Brukeren har mulighet til å zoome inn og ut enkelte porter, se figur 9.7. Digital Works mangler denne funksjonaliteten.

9.8 Usynlige porter og elementer

Brukeren har ikke mulighet til å flytte porter og andre komponenter med mus eller tastatur over til usynlige områder i Digital Circuit. I Digital Works kan vi flytte porten med mus/tastatur under menyraden, hvor det ikke er mulig å hente tilbake. På figuren 9.8 vises det at DigitalWorks tar ikke hensyn til dette.



Figur 9.7: Zoom in/ut mulighet



Figur 9.8: Usynlige porter i DigitalWorks

9.9 Vurdering

Digital Circuit er etter min oppfatning blitt en program som er intuitivt, lett å forstå og arbeide med. Utifra punktene over ser vi at Digital Circuit har fordeler framfor Digital Works. Digital Circuit oppfører seg etter min mening mer logisk riktig, og lar ikke brukeren utføre funksjoner som bryter med regler som gjelder ved tegning av digitale kretser. Digital Circuit er etter min mening mer intuitivt, lettere å forstå og arbeide med og et velegnet verktøy til å designe enkle digitale kretser med.

I prosjektet er nesten alle de kravene som ble stilt til programmet i startfasen oppnådd, se side 3. Det viktigste som står igjen er markroer og tilbakekoblinger. Applikasjonen må også utvides med flere funksjoner slik at man kan designe flere typer kretser med markroer og vipper.

Programmeringsspråket Java er brukt til utviklingen, og det har vist seg at det er rask nok til en interaktiv applikasjon som denne. Det har også de grafiske mulighetene som trengs. Etter min mening har Java vist seg til å være riktig språk å utvikle denne applikasjonen med.

Selv om det påpekes mange fordeler ved Digital Circuit, betyr det ikke at denne applikasjonen er feilfri. Applikasjonen må videreutvikles og tilpasses for at høgskoler og universiteter kan få full nytte av dette programmet.

Bibliografi

[1] Dr. Salah Adam. Threads, 2008. http://faculty.kfupm.edu.sa/ICS/adam/ICS431/Chap04.ppt#337,18,--%20Many-to-Many%20Model Aksessert den 18 januar 2009.

[2] Anonymous. Threads and swing, 2000. http://java.sun.com/products/jfc/tsc/articles/threads/threads1.html Aksessert den 18 januar 2009.

[3] Anonymous. Component, 2003. http://java.sun.com/j2se/1.4.2/docs/api/java/awt/Component.html Aksessert den 18 januar 2009.

[4] Anonymous. Jcomponent, 2003. http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/JComponent.html Aksessert den 18 januar 2009.

[5] Anonymous. Java - green threads in java, 2004. http://www.velocityreviews.com/forums/t130194-green-threads-in-java.html Aksessert den 18 januar 2009.

[6] Anonymous. User/kernel level threads, 2004. http://groups.google.com/group/comp.os.linux.development.system/browse_thread/thread/b2d3b08d2996d5d9/eb12ac41d07e3b6d?lnk=st&q=threads&rnum=1&hl=en#eb12ac41d07e3b6d Aksessert den 18 januar 2009.

[7] Anonymous. Adder, 2007. http://en.wikipedia.org/wiki/Half_adder Aksessert den 24 november 2008.

[8] Anonymous. Component-based software engineering, 2007. http://en.wikipedia.org/wiki/Sprite_(computer_graphics) Aksessert den 10 januar 2008.

[9] Anonymous. Component-based software engineering, 2007. http://en.wikipedia.org/wiki/Software_componentry Aksessert den 03 desember 2007.

[10] Anonymous. Java plattform og programmeringsspråk, 2007. http://www.tip.no/tini/3_java_teknologi.asp Aksessert den 18 januar 2009.

[11] Anonymous. Container : Java glossary, 2008. http://mindprod.com/jgloss/container.html Aksessert den 18 januar 2009.

[12] Anonymous. Datastruktur, 2008. http://no.wikipedia.org/wiki/Datastruktur Aksessert den 18 januar 2009.

[13] Anonymous. The jcomponent class, 2008. http://java.sun.com/docs/books/tutorial/uiswing/components/jcomponent.html Aksessert den 18 januar 2009.

[14] Anonymous. Lesson: Concurrency, 2008. http://java.sun.com/docs/books/tutorial/essential/concurrency/ Aksessert den 18 januar 2009.

[15] Anonymous. Selecting a font, 2008. http://java.sun.com/docs/books/tutorial/2d/text/fonts.html Aksessert den 20 mai 2008.

[16] Anonymous. The history of threads, 2009. http://www.faqs.org/faqs/os-research/part1/section-10.html Aksessert den 18 januar 2009.

[17] Anonymous. Invoking the java virtual machine, 2009. http://www.iam.ubc.ca/guides/javatut99/native1.1/invoking/invo.html Aksessert den 18 januar 2009.

[18] Anonymous. Threading, 2009. http://72.5.124.55/docs/hotspot/threads/threads.html Aksessert den 18 januar 2009.

[19] Anonymous. Thread (computer science), 2009. http://en.wikipedia.org/wiki/Thread_(computer_science) Aksessert den 18 januar 2009.

[20] Anonymous. Thread : Java glossary, 2009. http://mindprod.com/jgloss/thread.html Aksessert den 18 januar 2009.

[21] Anonymous. Tråder i java, 2009. http://www.idi.ntnu.no/emner/tdt4190/programvare/traader.html Aksessert den 18 januar 2009.

[22] Anonymous. And gate, undated. http://en.wikipedia.org/wiki/And_gate Aksessert den 06 november 2007.

[23] Anonymous. Common public license, undated. http://en.wikipedia.org/wiki/Common_Public_License Aksessert den 27 september 2007.

[24] Anonymous. Eclipse public license, undated. http://en.wikipedia.org/wiki/Eclipse_Public_License Aksessert den 27 september 2007.

[25] Anonymous. Eclipse (software), undated. http://en.wikipedia.org/wiki/Eclipse_%28software%29 Aksessert den 27 september 2007.

[26] Anonymous. Gnu general public license, undated. http://en.wikipedia.org/wiki/GNU_General_Public_License Aksessert den 27 september 2007.

[27] Anonymous. Ibm visualage, undated. http://en.wikipedia.org/wiki/VisualAge Aksessert den 27 september 2007.

[28] Anonymous. Jbuilder, undated. http://en.wikipedia.org/wiki/Jbuilder Aksessert den 27 september 2007.

[29] Anonymous. Learning java 2d, part 1, undated. http://java.sun.com/developer/technicalArticles/GUI/java2d/java2dpart1.html Aksessert den 01 november 2007.

[30] Anonymous. Migrating to eclipse: A developer's guide to evaluating eclipse vs. jbuilder, undated. http://www.ibm.com/developerworks/library/os-ecjbuild/?ca=dgr-lnxw01JBuilder2Eclipse Aksessert den 27 september 2007.

[31] Anonymous. New java 2d features in j2se 5.0, undated. http://java.sun.com/j2se/1.5.0/docs/guide/2d/new_features.html Aksessert den 03 desember 2007.

[32] Anonymous. Tråder i java. http://www.idi.ntnu.no/emner/tdt4190/programvare/traader.html Aksessert den 10 januar 2008.

[33] David Austin. Beginning graphics: an introduction to java2d, 2007. http://merganser.math.gvsu.edu/david/reed05/lectures/graphics.html Aksessert den 20 mai 2008.

[34] ramesh bh. What are heavy weight and light weight components in java??, 2007. http://www.blurtit.com/q132749.html Aksessert den 20 mai 2008.

[35] Andrew Davison. *Killer Game Programming*. OREILLY, first utgave, 2005.

[36] David Flanagan. Java foundation classes in a nutshell, 1999. http://www.unix.org.ua/orelly/java-ent/jfc/ch02_01.htm#jfcnut-ch-2-ex-1 Aksessert den 20 mai 2008.

[37] Marty Hall. A quick tutorial for awt programmers, 1999. http://www.apl.jhu.edu/~hall/java/Swing-Tutorial/ Aksessert den 20 mai 2008.

[38] B.L. Hannah. Game design the object oriented way. http://www.csharpfriends.com/Articles/getArticle.aspx?articleID=387 Aksessert den 27 november 2007.

[39] Jalex. Array vs arraylist vs linkedlist vs vector, 2007. http://www.javafaq.nu/java-article1111.html Aksessert den 18 januar 2009.

[40] Jason Marshall. Multi-cpu vm support under windows?, 1999. http://archives.java.sun.com/cgi-bin/wa?A2=ind9902&L=rmi-users&P=46418 Aksessert den 18 januar 2009.

[41] Eirik Maus. Oppgaver med java threads, 14. oktober 1998. http://www.ifi.uio.no/inods/Gamle/H1998/oppgaver/parallelloppgaver.html Aksessert den 10 januar 2008.

[42] Eirik Maus. Oppgaver med java threads, 1998. http://www.ifi.uio.no/inods/Gamle/H1998/oppgaver/parallelloppgaver.html Aksessert den 18 januar 2009.

[43] Professor Mads Nygård. Synkronisering i java, 2006. http://www.idi.ntnu.no/emner/tdt4190/programvare/synkronisering.html Aksessert den 18 januar 2009.

[44] John Ousterhout. Why threads are a bad idea (for most purposes), 2008. http://www.softpanorama.org/People/Ousterhout/Threads/index.shtml Aksessert den 18 januar 2009.

[45] Alexander Potochkin's. Jxtransformer: The power of a real swing !, 2006. http://weblogs.java.net/blog/alexfromsun/archive/2006/07/jxtransformer_t.html Aksessert den 18 januar 2009.

[46] A Sun Developer Network Site. Using layout managers, 2008. http://java.sun.com/docs/books/tutorial/uiswing/layout/using.html#choosing Aksessert den 20 mai 2008.

[47] Rachel Struthers. Java 2d and svg ppt presentation, 2008. http://www.authorstream.com/Presentation/TSBaG-1129-java-2d-and-svg-scalable-vector-graphics-raster-java2d-education-powerpoint/ Aksessert den 18 januar 2009.

[48] Fred Swartz. Java: Summary: Graphics usage, 2006. http://www.leepoint.net/notes-java/summaries/summary-graphics-usage.html Aksessert den 15 november 2007.

[49] Bill Venners. How the java virtual machine performs thread synchronization, 1997. http://www.javaworld.com/javaworld/jw-07-1997/jw-07-hood.html Aksessert den 18 januar 2009.

[50] Sermet Yucel. What does swing is slow mean? http://www.javalobby.org/articles/swing_slow/index.jsp Aksessert den 03 desember 2007.

Del IV

Vedlegg

Kapittel 10

Programkoden

10.1 Elements.java

```
1  package gates;
2
3  import javax.swing.JComponent;
4
5  /**
6   * Super-klassen for alle komponenter i Digital Circuit.
7   *
8   * @author Kandeeban Arumugam
9   *
10  */
11 public class Elements extends JComponent{
12 }
```

10.2 Macro.java

```
1  package gates;
2
3  import gui.DrawPanel;
4  import java.awt.Color;
5  import java.awt.Component;
6  import java.awt.Graphics;
7  import java.awt.Graphics2D;
8  import java.awt.Point;
9  import java.awt.RenderingHints;
10 import java.awt.event.MouseEvent;
11 import java.awt.event.MouseListener;
12 import java.awt.event.MouseMotionListener;
13 import java.awt.geom.Rectangle2D;
14 import java.util.LinkedList;
15 import javax.swing.SwingUtilities;
16
17 /**
```

```
18  * Macro.java
19  *
20  * @author Kandeeban Arumugam
21  *
22  */
23 public class Macro extends Elements implements MouseListener,
24                          MouseMotionListener{
25   private Point firstClick;
26   Rectangle2D rec;
27   int x, y;
28   Macro selected;
29   LinkedList<Output> outList;
30   LinkedList<Input> inpList;
31   int outWid;
32
33   /**
34    * Konstruktøren tar imot panelet som inneholder kretsen
35    *
36    * @param pane
37    */
38   public Macro(DrawPanel pane)
39   {
40     this.setLayout(null);
41     selected = null;
42     x = 20;
43     y = 0;
44     outList = new LinkedList<Output>();
45     inpList = new LinkedList<Input>();
46
47     outWid = 20*2;
48
49     //rec = new Rectangle2D.Double(x,y,100,100);
50
51     getComp(pane.getComponents());
52     this.setSize((int)rec.getBounds().getWidth()+1+outWid,
53                  (int)rec.getBounds().getHeight()+1);
54     this.addMouseListener(this);
55     this.addMouseMotionListener(this);
56   }
57
58
59   public void getComp(Component[] com)
60   {
61     int antInter = 0;
62     int antLed = 0;
63     for(int i = 0; i < com.length; i++)
64     {
65       if(com[i] instanceof InteractiveInput)
66       {
67         antInter++;
68       }
69       else if(com[i] instanceof Led)
70       {
71         antLed++;
```

```
72          }
73
74      }
75
76      int max = 0;
77      if(antInter < antLed)
78      {
79        max = antLed;
80      }
81      else{
82        max = antInter;
83      }
84
85      int height = 15*max;
86      int width = 50;
87      rec = new Rectangle2D.Double(x,y,width,height);
88
89      int inS = height/antInter;
90      int insStart = inS/2;
91
92      int ouS = height/antLed;
93      int ousStart = ouS/2;
94
95      for(int i = 0; i < com.length; i++)
96      {
97        if(com[i] instanceof InteractiveInput)
98        {
99          Input in = new Input(20);
100
101         in.setBounds(0,insStart - in.getLinePointY(),
102         (int)in.getSize().getWidth(), (int)in.getSize().getHeight());
103         insStart += inS;
104         this.add(in);
105         InteractiveInput intera = (InteractiveInput) com[i];
106         in.neste = intera.getOut().neste;
107         inpList.add(in);
108       }
109       else if(com[i] instanceof Led)
110       {
111         Output out = new Output(20);
112         out.setBounds(width+ (int)out.getSize().getWidth(),
113         ousStart - out.getLinePointY(),
114             (int)out.getSize().getWidth(),
115             (int)out.getSize().getHeight());
116
117         ousStart += ouS;
118         this.add(out);
119
120         Led le = (Led) com[i];
121         le.getInput().forrige.neste = out;
122         outList.add(out);
123       }
124
125     }
```

```
126
127
128     }
129
130     public void paintComponent(Graphics g) {
131       Graphics2D g2 = (Graphics2D) g;
132       g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
133           RenderingHints.VALUE_ANTIALIAS_ON);
134       g2.setPaint(Color.WHITE);
135       //g2.setPaint(Color.yellow);
136       //g2.fillRect(0, 0, getWidth(), getHeight());
137       // g2.setPaint(Color.green);
138       g2.fill(rec);
139       g2.setPaint(Color.BLACK);
140       g2.draw(rec);
141
142       //this.setSize(rec.getBounds().getSize());
143     }
144
145 //   public boolean contains(int input_x, int input_y) {
146 //     return rec.contains(input_x, input_y);
147 //   }
148
149
150     @Override
151     public void mouseDragged(MouseEvent e) {
152       // TODO Auto-generated method stub
153       DrawPanel parent = (DrawPanel) this.getParent();
154       if (selected instanceof Macro && !parent.isWireModus()) {
155
156         Point mouse = e.getPoint();
157         Point new_m = SwingUtilities.convertPoint(this, mouse, parent);
158
159         int xm1 = (int) firstClick.getX();
160         int ym1 = (int) firstClick.getY();
161
162         int xm2 = (int) new_m.getX();
163         int ym2 = (int) new_m.getY();
164
165         int yd = (int) (ym2 - ym1);
166         int xd = (int) (xm2 - xm1);
167
168         if (canDrag(xd, yd)) {
169           if (parent.isSnapToGrid()) {
170             int gridFreq = parent.getGridFrequency();
171             Point loc = this.getLocation();
172             double xG = ((loc.getX() + xd) / gridFreq);
173             double yG = ((loc.getY() + yd) / gridFreq);
174
175             int xxG = (int) Math.round(xG) * gridFreq;
176             int yyG = (int) Math.round(yG) * gridFreq;
177             //moveGateGrid(xxG, yyG);
178             firstClick = new Point(xxG, yyG);
179           } else {
```

```
180             moveGatePixel(xd, yd);
181             firstClick = new_m;
182           }
183         }
184       }
185
186     }
187
188     public void moveGatePixel(int xd, int yd) {
189       Point loc = this.getLocation();
190       selected.setLocation((int) loc.getX() + xd, (int) loc.getY() + yd);
191       DrawPanel pane = (DrawPanel) this.getParent();
192
193       moveOutputWires();
194       moveInputWires();
195
196       pane.updateScrollBars(this);
197       pane.setSaveChanged(true);
198     }
199
200     public void moveGateGrid(int x, int y) {
201       selected.setLocation(x, y);
202       DrawPanel pane = (DrawPanel) this.getParent();
203
204       moveOutputWires();
205       moveInputWires();
206
207       pane.setSaveChanged(true);
208     }
209
210     public boolean canDrag(int xd, int yd) {
211       Point loc = this.getLocation();
212       int xPoint = (int) (loc.getX() + xd);
213       int yPoint = (int) (loc.getY() + yd);
214       if (xPoint > 0 && yPoint > 0) {
215         return true;
216       } else {
217         return false;
218       }
219
220     }
221
222     public void moveOutputWires() {
223       DrawPanel pane = (DrawPanel) this.getParent();
224
225       for(int i = 0; i < outList.size(); i++)
226       {
227       Output out = outList.get(i);
228       WireComponent wireC = (WireComponent) out.neste;
229       if (wireC != null) {
230         if (out.getUpdatePNr() == 1) {
231           Point p = out.getConnectionPoint(pane);
232           Point p2 = wireC.getParentP2();
233           wireC.setWireLine(p, p2);
```

```
234           // pane.setWireElement(wireC, p);
235         } else {
236           Point p = wireC.getParentP1();
237           Point p2 = out.getConnectionPoint(pane);
238           wireC.setWireLine(p, p2);
239           // pane.setWireElement(wireC, p);
240         }
241       }
242       }
243
244     }
245
246     public void moveInputWires() {
247       DrawPanel pane = (DrawPanel) this.getParent();
248       for (int i = 0; i < inpList.size(); i++) {
249         Input inp = inpList.get(i);
250         WireComponent wireCI = (WireComponent) inp.forrige;
251         if (wireCI != null) {
252           if (inp.getUpdatePNr() == 1) {
253             Point p = inp.getConnectionPoint(pane);
254             Point p2 = wireCI.getParentP2();
255             wireCI.setWireLine(p, p2);
256             // pane.setWireElement(wireCI, p);
257
258           } else {
259
260             Point p = wireCI.getParentP1();
261             Point p2 = inp.getConnectionPoint(pane);
262             wireCI.setWireLine(p, p2);
263             // pane.setWireElement(wireCI, p);
264           }
265         }
266
267         WireComponent wireCI2 = (WireComponent) inp.neste;
268
269         if (wireCI2 != null) {
270           if (inp.getUpdatePNr() == 1) {
271             Point p = inp.getConnectionPoint(pane);
272             Point p2 = wireCI2.getParentP2();
273             wireCI2.setWireLine(p, p2);
274             // pane.setWireElement(wireCI2, p);
275           } else {
276             Point p = wireCI2.getParentP1();
277             Point p2 = inp.getConnectionPoint(pane);
278             wireCI2.setWireLine(p, p2);
279             // pane.setWireElement(wireCI2, p);
280           }
281         }
282
283       }
284     }
285     @Override
286     public void mouseMoved(MouseEvent e) {
287       // TODO Auto-generated method stub
```

```
288
289     }
290
291
292     @Override
293     public void mouseClicked(MouseEvent e) {
294       // TODO Auto-generated method stub
295
296     }
297
298
299     @Override
300     public void mouseEntered(MouseEvent e) {
301       // TODO Auto-generated method stub
302
303     }
304
305
306     @Override
307     public void mouseExited(MouseEvent e) {
308       // TODO Auto-generated method stub
309
310     }
311
312
313     @Override
314     public void mousePressed(MouseEvent e) {
315       // TODO Auto-generated method stub
316       System.out.println("pressef");
317       DrawPanel parent = (DrawPanel) this.getParent();
318       if (!parent.isWireModus()) {
319         Point m = e.getPoint();
320         firstClick = SwingUtilities.convertPoint(this, m, parent);
321       }
322
323       selected = this;
324     }
325
326
327     /**
328      * Følgende metoder brukes ikke.
329      * Man kan unngå å ha med disse tomme metoder
330      * ved å arve klassen MouseAdapter.
331      */
332     public void mouseReleased(MouseEvent e) {
333       // TODO Auto-generated method stub
334
335     }
336 }
```

10.3 MovePoint.java

```
1  package gates;
2
3  import gui.DrawPanel;
4
5  import java.awt.Color;
6  import java.awt.Graphics;
7  import java.awt.Graphics2D;
8  import java.awt.Point;
9  import java.awt.event.MouseEvent;
10 import java.awt.event.MouseListener;
11 import java.awt.event.MouseMotionListener;
12 import java.awt.geom.Rectangle2D;
13
14 import javax.swing.SwingUtilities;
15
16 /**
17  * MovePoint.java
18  *
19  * @author Kandeeban Arumugam
20  *
21  */
22 public class MovePoint extends Elements implements MouseListener,
23                                 MouseMotionListener{
24   private Rectangle2D rec;
25   private WireComponent wire1, wire2;
26   private int updateP1, updateP2;
27   private Point firstClick;
28   private int size;
29
30   /**
31    * Konstruktøren
32    */
33   public MovePoint()
34   {
35     size = 7;
36     updateP1 = updateP2 = 0;
37     rec = new Rectangle2D.Double(0,0,size,size);
38     this.setSize((int)rec.getWidth() + 1,(int)rec.getHeight() + 1);
39     this.addMouseListener(this);
40     this.addMouseMotionListener(this);
41   }
42
43   public void paintComponent(Graphics g) {
44     Graphics2D g2 = (Graphics2D) g;
45     //g2.setPaint(Color.yellow);
46     //g2.fillRect(0, 0, 100, 100);
47     //g2.fill(rec);
48     g2.setPaint(Color.BLACK);
49     g2.draw(rec);
50   }
51
52   public void setWire1(WireComponent wi, int p)
53   {
```

```
54      this.wire1 = wi;
55      updateP1 = p;
56    }
57
58    public void setWire2(WireComponent wi, int p)
59    {
60      this.wire2 = wi;
61      this.updateP2 = p;
62    }
63
64    public WireComponent getWire1()
65    {
66      return wire1;
67    }
68
69    public WireComponent getWire2()
70    {
71      return wire2;
72    }
73
74    public void mouseDragged(MouseEvent e) {
75      // TODO Auto-generated method stub
76      Point mouse = e.getPoint();
77      Point new_m = SwingUtilities.convertPoint(this, mouse, this
78          .getParent());
79
80      int xm1 = (int) firstClick.getX();
81      int ym1 = (int) firstClick.getY();
82
83      int xm2 = (int) new_m.getX();
84      int ym2 = (int) new_m.getY();
85
86      int yd = (int) (ym2 - ym1);
87      int xd = (int) (xm2 - xm1);
88      Point loc = this.getLocation();
89
90      this.setLocation((int) loc.getX() + xd, (int) loc.getY() + yd);
91      firstClick = new_m;
92
93
94        DrawPanel pane = (DrawPanel) this.getParent();
95        Point p, p2;
96        if(updateP1 == 1)
97        {
98          p = new Point(
99              (int) (wire1.getParentP1().getX() + xd),
100             (int) (wire1.getParentP1().getY() + yd));
101         p2 = new Point((int) (wire1.getParentP2().getX()),
102             (int) (wire1.getParentP2().getY()));
103         wire1.setWireLine(p, p2);
104         pane.setWireElement(wire1, p);
105       }
106       else
107       {
```

```
108           p = new Point((int) (wire1.getParentP1().getX()),
109               (int) (wire1.getParentP1().getY()));
110           p2 = new Point(
111               (int) (wire1.getParentP2().getX() + xd),
112               (int) (wire1.getParentP2().getY() + yd));
113           wire1.setWireLine(p, p2);
114           pane.setWireElement(wire1, p);
115         }
116
117         if(updateP2 == 1)
118         {
119           p = new Point(
120               (int) (wire2.getParentP1().getX() + xd),
121               (int) (wire2.getParentP1().getY() + yd));
122           p2 = new Point((int) (wire2.getParentP2().getX()),
123               (int) (wire2.getParentP2().getY()));
124           wire2.setWireLine(p, p2);
125           pane.setWireElement(wire2, p);
126         }
127         else
128         {
129           p = new Point((int) (wire2.getParentP1().getX()),
130               (int) (wire2.getParentP1().getY()));
131           p2 = new Point(
132               (int) (wire2.getParentP2().getX() + xd),
133               (int) (wire2.getParentP2().getY() + yd));
134           wire2.setWireLine(p, p2);
135           pane.setWireElement(wire2, p);
136         }
137
138         pane.repaint();
139     }
140
141     public void mousePressed(MouseEvent e) {
142       Point m = e.getPoint();
143       firstClick = SwingUtilities.convertPoint(this, m, this.getParent());
144     }
145
146     /**
147      * Følgende metoder brukes ikke.
148      * Man kan unngå å ha med disse tomme metoder
149      * ved å arve klassen MouseAdapter.
150      */
151     public void mouseReleased(MouseEvent e) {
152     }
153     public void mouseMoved(MouseEvent e) {
154     }
155
156     public void mouseClicked(MouseEvent e) {
157     }
158
159     public void mouseEntered(MouseEvent e) {
160     }
161     public void mouseExited(MouseEvent e) {
```

```
162     }
163
164 }
```

10.4 Gate.java

```
1  package gates;
2
3  import gui.DrawPanel;
4
5  import java.awt.Color;
6  import java.awt.Dimension;
7  import java.awt.Graphics;
8  import java.awt.Graphics2D;
9  import java.awt.Point;
10 import java.awt.Rectangle;
11 import java.awt.RenderingHints;
12 import java.awt.event.ActionEvent;
13 import java.awt.event.ActionListener;
14 import java.awt.event.KeyEvent;
15 import java.awt.event.KeyListener;
16 import java.awt.event.MouseEvent;
17 import java.awt.event.MouseListener;
18 import java.awt.event.MouseMotionListener;
19 import java.awt.geom.GeneralPath;
20 import java.awt.geom.PathIterator;
21 import java.awt.geom.Rectangle2D;
22 import java.util.HashMap;
23 import javax.swing.JMenuItem;
24 import javax.swing.JPopupMenu;
25 import javax.swing.SwingUtilities;
26
27 /**
28  * Gate.java
29  * Super-klassen for alle portene i Digital Circuit.
30  *
31  * @author Kandeeban Arumugam
32  *
33  */
34 public abstract class Gate extends Elements implements MouseListener,
35     MouseMotionListener, KeyListener {
36
37   //Diverse variablene som brukes av porter.
38   protected GeneralPath path = new GeneralPath();
39   private Gate selectedGate;
40   protected float x, y;
41
42   protected int compSize, antallInput, output_Signal;
43   protected Input in[] = new Input[4];
44   protected Output out;
45   int xf, yf;
```

```
46    boolean moveMode, focussed;
47    private boolean selected;
48    private Point firstClick;
49    protected Dimension componentDim;
50    private int elementID;
51      JPopupMenu popMenu;
52      PopupListener popListener;
53      int lastIncomingRot;
54
55      //Brukes av rot-algoritmen
56      HashMap<Integer, Integer> rots;
57
58      /**
59       * Konstruktøren som definerer de felles
60       * egenskapene for en port.
61       */
62    public Gate() {
63
64      this.lastIncomingRot = -1;
65      this.elementID = 0;
66
67      xf = -1;
68      yf = -1;
69      output_Signal = -1;
70      selectedGate = null;
71      selected = false;
72      moveMode = false;
73      firstClick = new Point();
74
75      popMenu = new JPopupMenu();
76      popListener = new PopupListener(this);
77
78      JMenuItem item1 = new JMenuItem("Zoom +");
79      item1.setActionCommand("zoom+");
80      JMenuItem item2 = new JMenuItem("Zoom -");
81      item2.setActionCommand("zoom-");
82
83
84      item1.addActionListener(popListener);
85      item2.addActionListener(popListener);
86
87      popMenu.add(item1);
88      popMenu.add(item2);
89
90      this.setFocusable(true);
91      this.addMouseListener(this);
92      this.addMouseMotionListener(this);
93      this.addKeyListener(this);
94    }
95
96
97    /**
98     * Setter nøkkel-id'en.
99 ***
```

```
100 ␣␣␣*␣@param␣elementID
101 ␣␣␣*/
102 ␣␣public␣void␣setElementID(int␣elementID)␣{
103 ␣␣␣␣this.elementID␣=␣elementID;
104 ␣␣}
105
106 ␣␣/**
107 ␣␣␣*␣Returnerer␣nøkkel-id'en.
108     *
109     * @param elementID
110     */
111   public int getElementID() {
112     return this.elementID;
113   }
114
115
116   /**
117    * Selve <<Rot-algoritmen>>.
118    *
119    * Metoden som avgjør om hvilken av de innkommende
120    * tråder kan fortsette.
121    *
122    * @return
123    */
124   public boolean isReady() {
125     for (int j = 0; j < antallInput; j++) {
126       if (in[j].getConnected() && !in[j].isSignalRegistered()) {
127         return false;
128       }
129     }
130
131     int antallSig = -1;
132     rots = new HashMap<Integer, Integer>();
133     for(int j = 0; j < antallInput; j++)
134     {
135       if(in[j].rotAndForeldre.containsKey(lastIncomingRot))
136       {
137         if(antallSig == -1)
138         {
139           antallSig = in[j].rotAndForeldre.get(lastIncomingRot);
140         }
141         else
142         {
143           if(antallSig != in[j].rotAndForeldre.get(lastIncomingRot))
144           {
145             return false;
146           }
147         }
148
149       }
150       rots.putAll(in[j].rotAndForeldre);
151     }
152     return true;
153   }
```

```
154
155
156     /**
157      * Returnerer om antall innganger.
158      * @return
159      */
160     public int getAntallInput() {
161       return antallInput;
162     }
163
164     /**
165      * Returnerer inngangene.
166      * @return
167      */
168     public Input[] getInput() {
169       return in;
170     }
171
172     // overrides Component
173     public Rectangle getBounds() {
174       return path.getBounds();
175     }
176
177     // overrides JComponent
178     public void paintComponent(Graphics g) {
179       Graphics2D g2 = (Graphics2D) g;
180       g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
181           RenderingHints.VALUE_ANTIALIAS_ON);
182       g2.setPaint(Color.WHITE);
183       // g2.setPaint(Color.yellow);
184       // g2.fillRect(0, 0, getWidth(), getHeight());
185       // g2.setPaint(Color.green);
186       g2.fill(path);
187       g2.setPaint(Color.BLACK);
188       g2.draw(path);
189
190       if (selected) {
191         int size = 5;
192         Rectangle2D rec = new Rectangle2D.Double(0, 0, 5, 5);
193         Rectangle2D rec2 = new Rectangle2D.Double((int) this.getSize()
194             .getWidth()
195             - (size + 1), 0, size, size);
196         Rectangle2D rec3 = new Rectangle2D.Double((int) this.getSize()
197             .getWidth()
198             - (size + 1),
199             (int) this.getSize().getHeight() - (size + 1), size, size);
200         Rectangle2D rec4 = new Rectangle2D.Double(0, (int) this.getSize()
201             .getHeight()
202             - (size + 1), size, size);
203
204         g2.draw(rec);
205         g2.draw(rec2);
206         g2.draw(rec3);
207         g2.draw(rec4);
```

```
208
209       }
210       this.setSize(getSize());
211
212     }
213
214     /**
215      * Metoden definerer formen til komponenten.
216      *
217      * overrides JComponent
218      */
219     public boolean contains(int input_x, int input_y) {
220       if (out != null) {
221         int ou_x = out.getX();
222         int ou_y = out.getY();
223         int ou_w = out.getWidth();
224         int ou_h = out.getHeight();
225         if ((input_x <= (ou_x + ou_w) && input_x >= ou_x)
226             && (input_y <= (ou_y + ou_h) && input_y >= ou_y)) {
227           return true;
228         }
229       }
230
231       for (int i = 0; i < antallInput; i++) {
232         int in_x = in[i].getX();
233         int in_y = in[i].getY();
234         int in_w = in[i].getWidth();
235         int in_h = in[i].getHeight();
236         if ((input_x <= (in_x + in_w) && input_x >= in_x)
237             && (input_y <= (in_y + in_h) && input_y >= in_y)) {
238           return true;
239         }
240       }
241
242       PathIterator iter = path.getPathIterator(null);
243       double lastX = 0.0, lastY = 0.0;
244       while (!iter.isDone()) {
245         float[] coor = new float[6];
246         int type = iter.currentSegment(coor);
247         float x1 = coor[0];
248         float y1 = coor[1];
249         // float x2 = coor[2];
250         // float y2 = coor[3];
251         // float x3 = coor[4];
252         // float y3 = coor[5];
253
254         // f(x)= ax + b <--- rett linje
255         double a = 0.0, b = 0.0;
256         double y = 0.0;
257         switch (type) {
258         case PathIterator.SEG_MOVETO:
259           lastX = x1;
260           lastY = y1;
261           break;
```

```
262         case PathIterator.SEG_LINETO:
263           a = (y1 - lastY) / (x1 - lastX);
264           b = lastY - (a * lastX);
265           y = (a * input_x) + b;
266           if (y == input_y && (input_x >= lastX && input_x <= x1)) {
267             a = 0;
268             b = 0;
269             y = 0;
270             return true;
271           }
272
273           lastX = x1;
274           lastY = y1;
275         }
276         iter.next();
277
278       }
279       return path.contains(input_x, input_y);
280     }
281
282     // overrides Component
283     public Dimension getSize() {
284       return new Dimension((int) path.getBounds().getWidth(), path
285           .getBounds().height + 1);
286     }
287
288     public abstract void startSimulering();
289
290     public abstract void sendSignal();
291
292     public abstract void create();
293
294     public void setSelected(boolean value) {
295       DrawPanel parent = (DrawPanel) this.getParent();
296       if (value) {
297         parent.setCurrentSelectedGate(selectedGate);
298         selected = true;
299         this.requestFocus(true);
300       } else {
301         // parent.setCurrentSelectedGate(null);
302         selected = false;
303         this.requestFocus(false);
304       }
305       this.repaint();
306     }
307
308     public boolean isSelected() {
309       return selected;
310     }
311
312     private void showPopup(MouseEvent e) {
313       popMenu.show(e.getComponent(), e.getX(), e.getY());
314     }
315
```

```
316   public void mouseClicked(MouseEvent e) {
317     DrawPanel parent = (DrawPanel) this.getParent();
318     if (isSelected()) {
319       setSelected(false);
320     } else {
321       setSelected(true);
322     }
323     if (e.getButton() == MouseEvent.BUTTON3) {
324       if(parent.isCursorModus())
325       showPopup(e);
326     }
327   }
328
329   /**
330    * Metoden returnerer utgangen.
331    *
332    * @return
333    */
334   private Output getOutPut() {
335     return out;
336   }
337
338   /**
339    * Metoden brukes ikke.
340    */
341   public void mouseEntered(MouseEvent arg0) {
342   }
343
344   /**
345    * Metoden brukes ikke.
346    */
347   public void mouseExited(MouseEvent arg0) {
348   }
349
350   /**
351    * Metoden finner første trykk punktet.
352    * Sjekker om porten er allerede valgt, hvis ikke
353    * settes porten til å være valgt.
354    */
355   public void mousePressed(MouseEvent e) {
356     DrawPanel parent = (DrawPanel) this.getParent();
357     if (!parent.isWireModus()) {
358       Point m = e.getPoint();
359       firstClick = SwingUtilities.convertPoint(this, m, parent);
360       if (!isSelected()) {
361         selectedGate = (Gate) this;
362       }
363     }
364   }
365
366   /**
367    * Metoden brukes ikke.
368    */
369   public void mouseReleased(MouseEvent arg0) {
```

```
370   }
371
372   /**
373    * Metoden utfører flytting.
374    */
375   public void mouseDragged(MouseEvent e) {
376     DrawPanel parent = (DrawPanel) this.getParent();
377     if (selectedGate instanceof Gate && !parent.isWireModus()) {
378
379       Point mouse = e.getPoint();
380       Point new_m = SwingUtilities.convertPoint(this, mouse, parent);
381
382       int xm1 = (int) firstClick.getX();
383       int ym1 = (int) firstClick.getY();
384
385       int xm2 = (int) new_m.getX();
386       int ym2 = (int) new_m.getY();
387
388       int yd = (int) (ym2 - ym1);
389       int xd = (int) (xm2 - xm1);
390
391       if (canDrag(xd, yd)) {
392         if (parent.isSnapToGrid()) {
393           int gridFreq = parent.getGridFrequency();
394           Point loc = this.getLocation();
395           double xG = ((loc.getX() + xd) / gridFreq);
396           double yG = ((loc.getY() + yd) / gridFreq);
397
398           int xxG = (int) Math.round(xG) * gridFreq;
399           int yyG = (int) Math.round(yG) * gridFreq;
400           moveGateGrid(xxG, yyG);
401           firstClick = new Point(xxG, yyG);
402         } else {
403           moveGatePixel(xd, yd);
404           firstClick = new_m;
405         }
406       }
407     }
408   }
409
410   /**
411    * Metoden returnerer true, hvis brukeres prøver å
412    * flytte innenfor synlige området.
413    *
414    * @param xd
415    * @param yd
416    * @return
417    */
418   public boolean canDrag(int xd, int yd) {
419     Point loc = this.getLocation();
420     int xPoint = (int) (loc.getX() + xd);
421     int yPoint = (int) (loc.getY() + yd);
422     if (xPoint > 0 && yPoint > 0) {
423       return true;
```

```
424       } else {
425         return false;
426       }
427
428     }
429
430     /**
431      * Flytter porten med et gitt punkt.
432      *
433      * @param xd
434      * @param yd
435      */
436     public void moveGatePixel(int xd, int yd) {
437       Point loc = this.getLocation();
438       selectedGate.setLocation((int) loc.getX() + xd, (int) loc.getY() + yd);
439       DrawPanel pane = (DrawPanel) this.getParent();
440
441       moveOutputWires();
442       moveInputWires();
443
444       pane.updateScrollBars(this);
445       pane.setSaveChanged(true);
446     }
447
448     /**
449      * Metoden er delvis implementert for å utføre
450      * <<grid>>-funksjonaliteten.
451      *
452      * @param x
453      * @param y
454      */
455     public void moveGateGrid(int x, int y) {
456       selectedGate.setLocation(x, y);
457       DrawPanel pane = (DrawPanel) this.getParent();
458
459       moveOutputWires();
460       moveInputWires();
461
462       pane.setSaveChanged(true);
463     }
464
465     /**
466      * Metoden for å flytte utgangs-ledninger sammen med porten.
467      */
468     public void moveOutputWires() {
469       DrawPanel pane = (DrawPanel) this.getParent();
470       Output out = selectedGate.getOutPut();
471       WireComponent wireC = (WireComponent) out.neste;
472       if (wireC != null) {
473         if (out.getUpdatePNr() == 1) {
474           Point p = out.getConnectionPoint(pane);
475           Point p2 = wireC.getParentP2();
476           wireC.setWireLine(p, p2);
477           // pane.setWireElement(wireC, p);
```

```
478         } else {
479           Point p = wireC.getParentP1();
480           Point p2 = out.getConnectionPoint(pane);
481           wireC.setWireLine(p, p2);
482           // pane.setWireElement(wireC, p);
483         }
484       }
485
486     }
487
488     /**
489      * Metoden for å flytte inngangs-ledninger sammen med porten.
490      */
491     public void moveInputWires() {
492       DrawPanel pane = (DrawPanel) this.getParent();
493       int ant = selectedGate.getAntallInput();
494       Input[] inpt = selectedGate.getInput();
495
496       for (int i = 0; i < ant; i++) {
497         Input inp = inpt[i];
498         WireComponent wireCI = (WireComponent) inp.forrige;
499         if (wireCI != null) {
500           if (inp.getUpdatePNr() == 1) {
501             Point p = inp.getConnectionPoint(pane);
502             Point p2 = wireCI.getParentP2();
503             wireCI.setWireLine(p, p2);
504           } else {
505
506             Point p = wireCI.getParentP1();
507             Point p2 = inp.getConnectionPoint(pane);
508             wireCI.setWireLine(p, p2);
509           }
510         }
511
512         WireComponent wireCI2 = (WireComponent) inp.neste;
513
514         if (wireCI2 != null) {
515           if (inp.getUpdatePNr() == 1) {
516             Point p = inp.getConnectionPoint(pane);
517             Point p2 = wireCI2.getParentP2();
518             wireCI2.setWireLine(p, p2);
519           } else {
520             Point p = wireCI2.getParentP1();
521             Point p2 = inp.getConnectionPoint(pane);
522             wireCI2.setWireLine(p, p2);
523           }
524         }
525
526       }
527     }
528
529     /**
530      * Metoden sender videre events-ene til foreldre komponenten.
531      */
```

```
532     public void mouseMoved(MouseEvent e) {
533       DrawPanel parent = (DrawPanel) this.getParent();
534       parent.mouseMoved(e);
535     }
536 
537     /**
538      * Metoden brukes for å flytte porten med tastatur.
539      *
540      * @param xPixel
541      * @param yPixel
542      */
543     public void moveGateWithKeypad(int xPixel, int yPixel) {
544       if (canDrag(xPixel, yPixel)) {
545         moveGatePixel(xPixel, yPixel);
546       }
547     }
548 
549     /**
550      * Metoden tar imot alle tastatur events-ene.
551      */
552     public void keyPressed(KeyEvent evt) {
553       if (isSelected()) {
554         switch (evt.getKeyCode()) {
555         case KeyEvent.VK_LEFT: // move x coordinate left
556           moveGateWithKeypad(-1, 0);
557           break;
558         case KeyEvent.VK_RIGHT: // move x coordinate right
559           moveGateWithKeypad(1, 0);
560           break;
561         case KeyEvent.VK_UP: // move x coordinate right
562           moveGateWithKeypad(0, -1);
563           break;
564         case KeyEvent.VK_DOWN: // move x coordinate right
565           moveGateWithKeypad(0, 1);
566         }
567       }
568     }
569 
570     /**
571      * Metoden brukes ikke per idag.
572      */
573     public void keyReleased(KeyEvent evt) {
574     }
575 
576     /**
577      * Metoden brukes ikke per idag.
578      */
579     public void keyTyped(KeyEvent evt) {
580     }
581 
582     public abstract void updateNewSize();
583 
584     /**
585      * Klassen implementerer lytter for popup-menyen.
```

```
586     *
587     * @author Kandeeban Arumugam
588     */
589    class PopupListener implements ActionListener {
590      Gate parent;
591
592      public PopupListener(Gate pa) {
593        this.parent = pa;
594      }
595
596      public void actionPerformed(ActionEvent e) {
597        // TODO Auto-generated method stub
598          if (e.getActionCommand().equals("zoom+")) {
599            compSize += 5;
600            parent.updateNewSize();
601            parent.moveInputWires();
602            parent.moveOutputWires();
603
604          } else if (e.getActionCommand().equals("zoom-")) {
605            compSize -= 5;
606            parent.updateNewSize();
607            parent.moveInputWires();
608            parent.moveOutputWires();
609          }
610        }
611
612    }
613
614    /**
615     * Følgende metoder brukes ikke.
616     * Man kan unngå å ha med disse tomme metoder
617     * ved å arve klassen MouseAdapter.
618     */
619
620 }
```

10.5 AndGate.java

```
1  package gates;
2
3  import gui.DrawPanel;
4  import java.awt.Dimension;
5  import java.awt.Point;
6  import javax.swing.ButtonGroup;
7  import javax.swing.JCheckBoxMenuItem;
8  import javax.swing.JMenu;
9
10 /**
11  * AndGate.java
12  *
13  * @author Kandeeban Arumugam
```

```
14  */
15 public class AndGate extends Gate {
16   //Variablene brukes for å plassere utgangen
17   private int outX, outY;
18
19   /**
20    * Konstruktøren for AND-port.
21    *
22    * @param size - Størrelsen på porten.
23    * @param antallInp - Antall innganger til porten.
24    */
25   public AndGate(int size, int antallInp) {
26     this.setLayout(null);
27     this.x = size * 2;
28     this.y = size;
29     this.compSize = size;
30     this.antallInput = antallInp;
31
32     path.moveTo(x - compSize, y - compSize);
33     path.lineTo(x, y - compSize);
34     path.curveTo(x + ((compSize * 3) / 2), y - compSize, x
35         + ((compSize * 3) / 2), y + compSize, x, y + compSize);
36     path.lineTo(x - compSize, y + compSize);
37     path.closePath();
38
39     int inS = (int) (y * 2) / (antallInput);
40     int inSstart = inS / 2;
41
42     for (int i = 0; i < antallInput; i++) {
43       in[i] = new Input(compSize);
44       in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
45           .getSize().width, in[i].getSize().height);
46       inSstart += inS;
47       this.add(in[i]);
48     }
49     out = new Output(compSize);
50     outX = (int) (x + compSize + (compSize / 8));
51     outY = (int) y - out.getLinePointY();
52     out.setBounds(outX, outY, out.getSize().width, out.getSize().height);
53
54     this.add(out);
55     this.componentDim = new Dimension((int) outX + out.getBounds().width,
56         path.getBounds().height + 1);
57
58     makeInputPopupmenu(antallInput);
59   }
60
61   /**
62    * Denne metoden kan brukes til å gi mulighet for brukeren.
63    * slik at de kan selv bestemme antall innganger.
64    * @param antInput
65    */
66   public void makeInputPopupmenu(int antInput)
67   {
```

```
68      ButtonGroup group = new ButtonGroup();
69      JMenu inputSubMenu = new JMenu("Inputs");
70
71      //2
72      JCheckBoxMenuItem subMenuItem1 = new JCheckBoxMenuItem(antInput + "");
73      subMenuItem1.setSelected(true);
74      subMenuItem1.setActionCommand("inputs");
75      subMenuItem1.addActionListener(this.popListener);
76      inputSubMenu.add(subMenuItem1);
77      group.add(subMenuItem1);
78
79      //3
80      subMenuItem1 = new JCheckBoxMenuItem("3");
81      subMenuItem1.setSelected(true);
82      subMenuItem1.setActionCommand("inputs");
83      subMenuItem1.addActionListener(this.popListener);
84      inputSubMenu.add(subMenuItem1);
85      subMenuItem1.setEnabled(false);
86      group.add(subMenuItem1);
87
88      //4
89      subMenuItem1 = new JCheckBoxMenuItem("4");
90      subMenuItem1.setSelected(true);
91      subMenuItem1.setActionCommand("inputs");
92      subMenuItem1.addActionListener(this.popListener);
93      inputSubMenu.add(subMenuItem1);
94      subMenuItem1.setEnabled(false);
95      group.add(subMenuItem1);
96
97      popMenu.add(inputSubMenu);
98    }
99
100   /*******Metodene som brukes av XMLEncoder og XMLDecoder**********/
101
102   /**
103    * Tom konstrøktøren for AND-port.
104    *
105    * Denne metoden gjør ingenting, derfor må set-metodene brukes,
106    * til å angi nødvendige parametere.
107    *
108    * Deretter må create()-metoden kalles.
109    *
110    */
111   public AndGate() {
112   }
113
114
115   /**
116    * Denne metoden setteropp AND-porten, dersom nødvendige
117    * variablene er gitt ved å bruke set-metoder.
118    */
119   public void create() {
120     // TODO Auto-generated method stub
121     this.setLayout(null);
```

```
122       this.x = compSize * 2;
123       this.y = compSize;
124
125       path.moveTo(x - compSize, y - compSize);
126       path.lineTo(x, y - compSize);
127       path.curveTo(x + ((compSize * 3) / 2), y - compSize, x
128           + ((compSize * 3) / 2), y + compSize, x, y + compSize);
129       path.lineTo(x - compSize, y + compSize);
130       path.closePath();
131
132       for (int i = 0; i < antallInput; i++) {
133         in[i].create();
134         this.add(in[i]);
135       }
136
137       out.create();
138       outX = (int) (x + compSize + (compSize / 8));
139       this.add(out);
140       this.componentDim = new Dimension((int) outX + out.getBounds().width,
141           path.getBounds().height + 1);
142
143     }
144
145     /**
146      * Denne metoden brukes til å forstørre porten.
147      */
148     public void updateNewSize()
149     {
150       this.x = compSize * 2;
151       this.y = compSize;
152       path.reset();
153
154       path.moveTo(x - compSize, y - compSize);
155       path.lineTo(x, y - compSize);
156       path.curveTo(x + ((compSize * 3) / 2), y - compSize, x
157           + ((compSize * 3) / 2), y + compSize, x, y + compSize);
158       path.lineTo(x - compSize, y + compSize);
159       path.closePath();
160
161       int inS = (int) (y * 2) / (antallInput);
162       int inSstart = inS / 2;
163
164       for (int i = 0; i < antallInput; i++) {
165         in[i].setCompSize(compSize);
166         in[i].updateNewSize();
167
168         in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
169             .getSize().width, in[i].getSize().height);
170         inSstart += inS;
171       }
172       out.setCompSize(compSize);
173       out.updateNewSize();
174
175       outX = (int) (x + compSize + (compSize / 8));
```

```
176       outY = (int) y - out.getLinePointY();
177       out.setBounds(outX, outY, out.getSize().width, out.getSize().height);
178
179
180       this.componentDim = new Dimension((int) outX + out.getBounds().width,
181           path.getBounds().height + 1);
182     }
183
184     /**
185      * Metoden tar imot utgangen for denne porten.
186      *
187      * @param out
188      */
189     public void setOut(Output out) {
190       this.out = out;
191     }
192
193     /**
194      * Metoden returnerer utgangen til denne porten.
195      *
196      * @return
197      */
198     public Output getOut() {
199       return this.out;
200     }
201
202     /**
203      * Metoden setter inngangene til denne porten.
204      *
205      * @param in
206      */
207     public void setIn(Input[] in) {
208       this.in = in;
209     }
210
211     /**
212      * Metoden returnerer inngangene til denne porten.
213      *
214      * @return
215      */
216     public Input[] getIn() {
217       return this.in;
218     }
219
220     /**
221      * Metoden kan brukes til å plassere porten.
222      *
223      * @param locPo
224      */
225     public void setLocPo(Point locPo) {
226       this.setLocation(locPo);
227     }
228
229     /**
```

```
230     * Metoden returnerer plasseringspunktet til porten.
231     *
232     * @return
233     */
234    public Point getLocPo() {
235      return this.getLocation();
236    }
237
238    /**
239     * Metoden kan brukes til å definere størrelsen.
240     *
241     * @param compSize
242     */
243    public void setCompSize(int compSize) {
244      this.compSize = compSize;
245    }
246
247    /**
248     * Metoden returnerer definerte størrelsen.
249     *
250     * @return
251     */
252    public int getCompSize() {
253      return this.compSize;
254    }
255
256    /**
257     * Metoden kan brukes til å sette antall innganger.
258     *
259     * @param antallInput
260     */
261    public void setAntallInput(int antallInput) {
262      this.antallInput = antallInput;
263    }
264
265    /**
266     * Returnerer antall innganger.
267     *
268     */
269    public int getAntallInput() {
270      return antallInput;
271    }
272
273    /**
274     * Returnerer størrelsen.
275     */
276    public Dimension getSize() {
277      return componentDim;
278    }
279
280    /**************** END *************/
281
282    /**
283     * Metoden for å sette igang simulasjon.
```

```
284     *
285     * Metoden overrides Gate-metoden.
286     */
287    public synchronized void startSimulering() {
288      boolean ready = isReady();
289      if (ready) {
290        int sig = in[0].signal;
291        for (int i = 1; i < antallInput; i++) {
292          sig *= in[i].signal;
293        }
294
295        output_Signal = sig;
296        sendSignal();
297      }
298    }
299
300    /**
301     * Metoden sender videre utgangs-signalet.
302     */
303    public void sendSignal() {
304      DrawPanel parent = (DrawPanel) this.getParent();
305      final int sleep = parent.getClockSpeed();
306
307      WireElement wis = this.out.neste;
308      while (wis != null) {
309        wis.registerRots(rots, lastIncomingRot);
310        wis.setSignal(output_Signal);
311        wis.setShowCurent(true);
312        Thread t = Thread.currentThread();
313        try {
314          t.sleep(sleep);
315        } catch (InterruptedException e) {
316          e.printStackTrace();
317        }
318
319        wis = wis.neste;
320      }
321    }
322
323
324  }
```

10.6 XNorGate.java

```
1  package gates;
2
3  import gui.DrawPanel;
4
5  import java.awt.Dimension;
6  import java.awt.Point;
7  import java.awt.Shape;
```

```
8  import java.awt.geom.Ellipse2D;
9
10 import javax.swing.ButtonGroup;
11 import javax.swing.JCheckBoxMenuItem;
12 import javax.swing.JMenu;
13
14 /**
15  * XNorGate.java
16  *
17  * @author Kandeeban Arumugam
18  */
19 public class XNorGate extends Gate {
20   private int outX;
21
22   /**
23    * Konstruktøren for XNorGate-port.
24    *
25    * @param size - Størrelsen på porten.
26    * @param antallInp - Antall innganger til porten.
27    */
28   public XNorGate(int størrelse, int antallInp) {
29     this.compSize = størrelse;
30     this.x = compSize * 2;
31     this.y = compSize;
32     this.antallInput = antallInp;
33     this.setLayout(null);
34
35     Shape circle = new Ellipse2D.Double(x + (compSize) - (compSize / 9.8),
36         y - (compSize / 4), (compSize / 2), (compSize / 2));
37
38     x -= 2;
39     path.moveTo(x - compSize - (compSize / 3) - (compSize / 5), y
40         + compSize);
41     path.quadTo(x - (compSize / 2.4) - (compSize / 5), y, x - compSize
42         - (compSize / 3) - (compSize / 5), y - compSize);
43     x += 2;
44     path.moveTo(x - compSize - (compSize / 3), y + compSize);
45     path.quadTo(x - (compSize / 2.4), y, x - compSize - (compSize / 3), y
46         - compSize);
47     path.lineTo(x - (compSize / 3), y - compSize);
48     path.curveTo(x + (compSize / 3), y - (compSize / 1.2),
49         x + compSize / 2, y - (compSize / 1.5), x + (compSize)
50             - (compSize / 9.8), y);
51     path.curveTo(x + compSize / 2, y + (compSize / 1.5),
52         x + (compSize / 3), y + (compSize / 1.2), x - (compSize / 3), y
53             + compSize);
54     path.closePath();
55
56     path.append(circle, false);
57
58     int inS = (int) (y * 2) / (antallInput);
59     int inSstart = inS / 2;
60
61     compSize -= 2;
```

```
62      for (int i = 0; i < antallInput; i++) {
63        in[i] = new Input(compSize - (compSize / 5));
64        in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
65            .getSize().width, in[i].getSize().height);
66        inSstart += inS;
67        this.add(in[i]);
68      }
69      compSize += 2;
70      out = new Output(compSize);
71      outX = circle.getBounds().x + circle.getBounds().width;
72
73      out.setBounds(outX, (int) y - out.getLinePointY(), out.getSize().width,
74          out.getSize().height);
75
76      this.add(out);
77      this.componentDim = new Dimension((int) outX + out.getBounds().width,
78          path.getBounds().height + 1);
79
80      makeInputPopupmenu(antallInput);
81    }
82
83    /**
84     * Denne metoden kan brukes til å gi mulighet for brukeren.
85     * slik at de kan selv bestemme antall innganger.
86     * @param antInput
87     */
88    public void makeInputPopupmenu(int antInput)
89    {
90      ButtonGroup group = new ButtonGroup();
91      JMenu inputSubMenu = new JMenu("Inputs");
92
93      //2
94      JCheckBoxMenuItem subMenuItem1 = new JCheckBoxMenuItem(antInput + "");
95      subMenuItem1.setSelected(true);
96      subMenuItem1.setActionCommand("inputs");
97      subMenuItem1.addActionListener(this.popListener);
98      inputSubMenu.add(subMenuItem1);
99      group.add(subMenuItem1);
100
101     //3
102     subMenuItem1 = new JCheckBoxMenuItem("3");
103     subMenuItem1.setSelected(true);
104     subMenuItem1.setActionCommand("inputs");
105     subMenuItem1.addActionListener(this.popListener);
106     inputSubMenu.add(subMenuItem1);
107     subMenuItem1.setEnabled(false);
108     group.add(subMenuItem1);
109
110     //4
111     subMenuItem1 = new JCheckBoxMenuItem("4");
112     subMenuItem1.setSelected(true);
113     subMenuItem1.setActionCommand("inputs");
114     subMenuItem1.addActionListener(this.popListener);
115     inputSubMenu.add(subMenuItem1);
```

```
116       subMenuItem1.setEnabled(false);
117       group.add(subMenuItem1);
118
119       popMenu.add(inputSubMenu);
120     }
121
122     /**Metodene som brukes av XMLEncoder og XMLDecoder ***/
123     /**
124      * Tom konstrøktøren for XNorGate-port.
125      *
126      * Denne metoden gjør ingenting, derfor må set-metodene brukes,
127      * til å angi nødvendige parametere.
128      *
129      * Deretter må create()-metoden kalles.
130      *
131      */
132     public XNorGate() {
133     }
134
135     /**
136      * Denne metoden setteropp XNorGate-porten, dersom nødvendige
137      * variablene er gitt ved å bruke set-metoder.
138      */
139     public void create() {
140       // TODO Auto-generated method stub
141       this.x = compSize * 2;
142       this.y = compSize;
143       this.setLayout(null);
144
145       Shape circle = new Ellipse2D.Double(x + (compSize) - (compSize / 9.8),
146           y - (compSize / 4), (compSize / 2), (compSize / 2));
147
148       x -= 2;
149       path.moveTo(x - compSize - (compSize / 3) - (compSize / 5), y
150           + compSize);
151       path.quadTo(x - (compSize / 2.4) - (compSize / 5), y, x - compSize
152           - (compSize / 3) - (compSize / 5), y - compSize);
153       x += 2;
154       path.moveTo(x - compSize - (compSize / 3), y + compSize);
155       path.quadTo(x - (compSize / 2.4), y, x - compSize - (compSize / 3), y
156           - compSize);
157       path.lineTo(x - (compSize / 3), y - compSize);
158       path.curveTo(x + (compSize / 3), y - (compSize / 1.2),
159           x + compSize / 2, y - (compSize / 1.5), x + (compSize)
160               - (compSize / 9.8), y);
161       path.curveTo(x + compSize / 2, y + (compSize / 1.5),
162           x + (compSize / 3), y + (compSize / 1.2), x - (compSize / 3), y
163               + compSize);
164       path.closePath();
165
166       path.append(circle, false);
167
168       for (int i = 0; i < antallInput; i++) {
169         in[i].create();
```

```
170         this.add(in[i]);
171       }
172       outX = circle.getBounds().x + circle.getBounds().width;
173       out.create();
174       this.add(out);
175       this.componentDim = new Dimension((int) outX + out.getBounds().width,
176           path.getBounds().height + 1);
177
178     }
179
180     /**
181      * Metoden tar imot utgangen for denne porten.
182      *
183      * @param out
184      */
185     public void setOut(Output out) {
186       this.out = out;
187     }
188
189     /**
190      * Metoden returnerer utgangen til denne porten.
191      *
192      * @return
193      */
194     public Output getOut() {
195       return this.out;
196     }
197
198     /**
199      * Metoden setter inngangene til denne porten.
200      *
201      * @param in
202      */
203     public void setIn(Input[] in) {
204       this.in = in;
205     }
206
207     /**
208      * Metoden returnerer inngangene til denne porten.
209      *
210      * @return
211      */
212     public Input[] getIn() {
213       return this.in;
214     }
215
216     /**
217      * Metoden kan brukes til å plassere porten.
218      *
219      * @param locPo
220      */
221     public void setLocPo(Point locPo) {
222       this.setLocation(locPo);
223     }
```

```
224
225     /**
226      * Metoden returnerer plasseringspunktet til porten.
227      *
228      * @return
229      */
230     public Point getLocPo() {
231       return this.getLocation();
232     }
233
234     /**
235      * Metoden kan brukes til å definere størrelsen.
236      *
237      * @param compSize
238      */
239     public void setCompSize(int compSize) {
240       this.compSize = compSize;
241     }
242
243     /**
244      * Metoden returnerer definerte størrelsen.
245      *
246      * @return
247      */
248     public int getCompSize() {
249       return this.compSize;
250     }
251
252     /**
253      * Metoden kan brukes til å sette antall innganger.
254      *
255      * @param antallInput
256      */
257     public void setAntallInput(int antallInput) {
258       this.antallInput = antallInput;
259     }
260
261     /**
262      * Returnerer antall innganger.
263      *
264      */
265     public int getAntallInput() {
266       return antallInput;
267     }
268
269     /**
270      * Returnerer størrelsen.
271      */
272     public Dimension getSize() {
273       return componentDim;
274     }
275
276     /************ END *********/
277     /**
```

```
278     * Metoden for å sette igang simulasjon.
279     *
280     * Metoden overrides Gate-metoden.
281     */
282    public synchronized void startSimulering() {
283
284      boolean ready = isReady();
285      if (ready) {
286        int sig = in[0].signal;
287        for (int i = 1; i < antallInput; i++) {
288          sig ^= in[i].signal;
289        }
290
291        if (sig == 0) {
292          output_Signal = 1;
293        } else {
294          output_Signal = 0;
295        }
296
297        sendSignal();
298      }
299    }
300
301    /**
302     * Metoden sender videre utgangs-signalet.
303     */
304    public void sendSignal() {
305      DrawPanel parent = (DrawPanel) this.getParent();
306      final int sleep = parent.getClockSpeed();
307
308      WireElement wis = this.out.neste;
309      while (wis != null) {
310        wis.registerRots(rots, lastIncomingRot);
311        wis.setSignal(output_Signal);
312        wis.setShowCurent(true);
313        Thread t = Thread.currentThread();
314        try {
315          t.sleep(sleep);
316        } catch (InterruptedException e) {
317          // TODO Auto-generated catch block
318          e.printStackTrace();
319        }
320        wis = wis.neste;
321      }
322    }
323
324    /**
325     * Denne metoden brukes til å forstørre porten.
326     */
327    public void updateNewSize() {
328      this.x = compSize * 2;
329      this.y = compSize;
330      path.reset();
331
```

```
332         Shape circle = new Ellipse2D.Double(x + (compSize) - (compSize / 9.8),
333             y - (compSize / 4), (compSize / 2), (compSize / 2));
334 
335         x -= 2;
336         path.moveTo(x - compSize - (compSize / 3) - (compSize / 5), y
337             + compSize);
338         path.quadTo(x - (compSize / 2.4) - (compSize / 5), y, x - compSize
339             - (compSize / 3) - (compSize / 5), y - compSize);
340         x += 2;
341         path.moveTo(x - compSize - (compSize / 3), y + compSize);
342         path.quadTo(x - (compSize / 2.4), y, x - compSize - (compSize / 3), y
343             - compSize);
344         path.lineTo(x - (compSize / 3), y - compSize);
345         path.curveTo(x + (compSize / 3), y - (compSize / 1.2),
346             x + compSize / 2, y - (compSize / 1.5), x + (compSize)
347                 - (compSize / 9.8), y);
348         path.curveTo(x + compSize / 2, y + (compSize / 1.5),
349             x + (compSize / 3), y + (compSize / 1.2), x - (compSize / 3), y
350                 + compSize);
351         path.closePath();
352 
353         path.append(circle, false);
354 
355         int inS = (int) (y * 2) / (antallInput);
356         int inSstart = inS / 2;
357 
358         compSize -= 2;
359         for (int i = 0; i < antallInput; i++) {
360           in[i].setCompSize(compSize - (compSize / 5));
361           in[i].updateNewSize();
362 
363           in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
364               .getSize().width, in[i].getSize().height);
365           inSstart += inS;
366         }
367         compSize += 2;
368         out.setCompSize(compSize);
369         out.updateNewSize();
370 
371         outX = circle.getBounds().x + circle.getBounds().width;
372 
373         out.setBounds(outX, (int) y - out.getLinePointY(), out.getSize().width,
374             out.getSize().height);
375 
376         this.componentDim = new Dimension((int) outX + out.getBounds().width,
377             path.getBounds().height + 1);
378 
379       }
380 
381     }
```

10.7 XOrGate.java

```
1  package gates;
2
3  import gui.DrawPanel;
4
5  import java.awt.Dimension;
6  import java.awt.Point;
7
8  import javax.swing.ButtonGroup;
9  import javax.swing.JCheckBoxMenuItem;
10 import javax.swing.JMenu;
11
12 /**
13  * XOrGate.java
14  *
15  * @author Kandeeban Arumugam
16  */
17 public class XOrGate extends Gate {
18   int outX;
19
20   /**
21    * Konstruktøren for XOrGate-port.
22    *
23    * @param size - Størrelsen på porten.
24    * @param antallInp - Antall innganger til porten.
25    */
26   public XOrGate(int size, int antallInp) {
27     this.compSize = size;
28     this.x = compSize * 2;
29     this.y = compSize;
30     this.antallInput = antallInp;
31     this.setLayout(null);
32
33     x -= 2;
34     path.moveTo(x - compSize - (compSize / 3) - (compSize / 5), y
35         + compSize);
36     path.quadTo(x - (compSize / 2.4) - (compSize / 5), y, x - compSize
37         - (compSize / 3) - (compSize / 5), y - compSize);
38     x += 2;
39     path.moveTo(x - compSize - (compSize / 3), y + compSize);
40     path.quadTo(x - (compSize / 2.4), y, x - compSize - (compSize / 3), y
41         - compSize);
42     path.lineTo(x - (compSize / 3), y - compSize);
43     path.curveTo(x + (compSize / 3), y - (compSize / 1.2),
44         x + compSize / 2, y - (compSize / 1.5), x + (compSize)
45             - (compSize / 9.8), y);
46     path.curveTo(x + compSize / 2, y + (compSize / 1.5),
47         x + (compSize / 3), y + (compSize / 1.2), x - (compSize / 3), y
48             + compSize);
49     path.closePath();
50
51     int inS = (int) (y * 2) / (antallInput);
```

```
52      int inSstart = inS / 2;
53
54      compSize -= 2;
55      for (int i = 0; i < antallInput; i++) {
56        in[i] = new Input(compSize - (compSize / 5));
57        in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
58            .getSize().width, in[i].getSize().height);
59        inSstart += inS;
60        this.add(in[i]);
61      }
62      compSize += 2;
63      out = new Output(compSize);
64      outX = (int) (x + (compSize) - (compSize / 9.8));
65
66      out.setBounds(outX, (int) y - out.getLinePointY(), out.getSize().width,
67          out.getSize().height);
68
69      this.add(out);
70      this.componentDim = new Dimension((int) outX + out.getBounds().width,
71          path.getBounds().height + 1);
72
73      makeInputPopupmenu(antallInput);
74    }
75
76    /**
77     * Denne metoden kan brukes til å gi mulighet for brukeren.
78     * slik at de kan selv bestemme antall innganger.
79     * @param antInput
80     */
81    public void makeInputPopupmenu(int antInput)
82    {
83      ButtonGroup group = new ButtonGroup();
84      JMenu inputSubMenu = new JMenu("Inputs");
85
86      //2
87      JCheckBoxMenuItem subMenuItem1 = new JCheckBoxMenuItem(antInput + "");
88      subMenuItem1.setSelected(true);
89      subMenuItem1.setActionCommand("inputs");
90      subMenuItem1.addActionListener(this.popListener);
91      inputSubMenu.add(subMenuItem1);
92      group.add(subMenuItem1);
93
94      //3
95      subMenuItem1 = new JCheckBoxMenuItem("3");
96      subMenuItem1.setSelected(true);
97      subMenuItem1.setActionCommand("inputs");
98      subMenuItem1.addActionListener(this.popListener);
99      inputSubMenu.add(subMenuItem1);
100     subMenuItem1.setEnabled(false);
101     group.add(subMenuItem1);
102
103     //4
104     subMenuItem1 = new JCheckBoxMenuItem("4");
105     subMenuItem1.setSelected(true);
```

```
106       subMenuItem1.setActionCommand("inputs");
107       subMenuItem1.addActionListener(this.popListener);
108       inputSubMenu.add(subMenuItem1);
109       subMenuItem1.setEnabled(false);
110       group.add(subMenuItem1);
111
112       popMenu.add(inputSubMenu);
113     }
114
115     /****** Metodene som brukes av XMLEncoder og XMLDecoder *******/
116     /**
117      * Tom konstrøktøren for XOrGate-port.
118      *
119      * Denne metoden gjør ingenting, derfor må set-metodene brukes,
120      * til å angi nødvendige parametere.
121      *
122      * Deretter må create()-metoden kalles.
123      *
124      */
125     public XOrGate() {
126     }
127
128     /**
129      * Denne metoden setteropp XOrGate-porten, dersom nødvendige
130      * variablene er gitt ved å bruke set-metoder.
131      */
132     public void create() {
133       // TODO Auto-generated method stub
134       this.x = compSize * 2;
135       this.y = compSize;
136       this.setLayout(null);
137
138       x -= 2;
139       path.moveTo(x - compSize - (compSize / 3) - (compSize / 5), y
140           + compSize);
141       path.quadTo(x - (compSize / 2.4) - (compSize / 5), y, x - compSize
142           - (compSize / 3) - (compSize / 5), y - compSize);
143       x += 2;
144       path.moveTo(x - compSize - (compSize / 3), y + compSize);
145       path.quadTo(x - (compSize / 2.4), y, x - compSize - (compSize / 3), y
146           - compSize);
147       path.lineTo(x - (compSize / 3), y - compSize);
148       path.curveTo(x + (compSize / 3), y - (compSize / 1.2),
149           x + compSize / 2, y - (compSize / 1.5), x + (compSize)
150               - (compSize / 9.8), y);
151       path.curveTo(x + compSize / 2, y + (compSize / 1.5),
152           x + (compSize / 3), y + (compSize / 1.2), x - (compSize / 3), y
153               + compSize);
154       path.closePath();
155
156       for (int i = 0; i < antallInput; i++) {
157         in[i].create();
158         this.add(in[i]);
159       }
```

```
160
161       outX = (int) (x + (compSize) - (compSize / 9.8));
162       out.create();
163       this.add(out);
164       this.componentDim = new Dimension((int) outX + out.getBounds().width,
165           path.getBounds().height + 1);
166
167     }
168
169     /**
170      * Metoden tar imot utgangen for denne porten.
171      *
172      * @param out
173      */
174     public void setOut(Output out) {
175       this.out = out;
176     }
177
178     /**
179      * Metoden returnerer utgangen til denne porten.
180      *
181      * @return
182      */
183     public Output getOut() {
184       return this.out;
185     }
186
187     /**
188      * Metoden setter inngangene til denne porten.
189      *
190      * @param in
191      */
192     public void setIn(Input[] in) {
193       this.in = in;
194     }
195
196     /**
197      * Metoden returnerer inngangene til denne porten.
198      *
199      * @return
200      */
201     public Input[] getIn() {
202       return this.in;
203     }
204
205     /**
206      * Metoden kan brukes til å plassere porten.
207      *
208      * @param locPo
209      */
210     public void setLocPo(Point locPo) {
211       this.setLocation(locPo);
212     }
213
```

```
214     /**
215      * Metoden returnerer plasseringspunktet til porten.
216      *
217      * @return
218      */
219     public Point getLocPo() {
220       return this.getLocation();
221     }
222 
223     /**
224      * Metoden kan brukes til å definere størrelsen.
225      *
226      * @param compSize
227      */
228     public void setCompSize(int compSize) {
229       this.compSize = compSize;
230     }
231 
232     /**
233      * Metoden returnerer definerte størrelsen.
234      *
235      * @return
236      */
237     public int getCompSize() {
238       return this.compSize;
239     }
240 
241     /**
242      * Metoden kan brukes til å sette antall innganger.
243      *
244      * @param antallInput
245      */
246     public void setAntallInput(int antallInput) {
247       this.antallInput = antallInput;
248     }
249 
250     /**
251      * Returnerer antall innganger.
252      *
253      */
254     public int getAntallInput() {
255       return antallInput;
256     }
257 
258     /**
259      * Returnerer størrelsen.
260      */
261     public Dimension getSize() {
262       return componentDim;
263     }
264 
265     /******** END ***************/
266     /**
267      * Metoden for å sette igang simulasjon.
```

```
268     *
269     * Metoden overrides Gate-metoden.
270     */
271    public synchronized void startSimulering() {
272
273      boolean ready = isReady();
274      if (ready) {
275        // TODO Auto-generated method stub
276        int sig = in[0].signal;
277        for (int i = 1; i < antallInput; i++) {
278          sig ^= in[i].signal;
279        }
280
281        output_Signal = sig;
282
283        sendSignal();
284      }
285    }
286
287    /**
288     * Metoden sender videre utgangs-signalet.
289     */
290    public void sendSignal() {
291      DrawPanel parent = (DrawPanel) this.getParent();
292      final int sleep = parent.getClockSpeed();
293
294      WireElement wis = this.out.neste;
295      while (wis != null) {
296        wis.registerRots(rots, lastIncomingRot);
297        wis.setSignal(output_Signal);
298        wis.setShowCurent(true);
299        Thread t = Thread.currentThread();
300        try {
301          t.sleep(sleep);
302        } catch (InterruptedException e) {
303          // TODO Auto-generated catch block
304          e.printStackTrace();
305        }
306        wis = wis.neste;
307      }
308    }
309
310
311    /**
312     * Denne metoden brukes til å forstørre porten.
313     */
314    public void updateNewSize() {
315      this.x = compSize * 2;
316      this.y = compSize;
317      path.reset();
318
319      x -= 2;
320      path.moveTo(x - compSize - (compSize / 3) - (compSize / 5), y
321          + compSize);
```

```
322         path.quadTo(x - (compSize / 2.4) - (compSize / 5), y, x - compSize
323             - (compSize / 3) - (compSize / 5), y - compSize);
324         x += 2;
325         path.moveTo(x - compSize - (compSize / 3), y + compSize);
326         path.quadTo(x - (compSize / 2.4), y, x - compSize - (compSize / 3), y
327             - compSize);
328         path.lineTo(x - (compSize / 3), y - compSize);
329         path.curveTo(x + (compSize / 3), y - (compSize / 1.2),
330             x + compSize / 2, y - (compSize / 1.5), x + (compSize)
331                 - (compSize / 9.8), y);
332         path.curveTo(x + compSize / 2, y + (compSize / 1.5),
333             x + (compSize / 3), y + (compSize / 1.2), x - (compSize / 3), y
334                 + compSize);
335         path.closePath();
336 
337         int inS = (int) (y * 2) / (antallInput);
338         int inSstart = inS / 2;
339 
340         compSize -= 2;
341         for (int i = 0; i < antallInput; i++) {
342           in[i].setCompSize(compSize - (compSize / 5));
343           in[i].updateNewSize();
344 
345           in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
346               .getSize().width, in[i].getSize().height);
347           inSstart += inS;
348         }
349         compSize += 2;
350         out.setCompSize(compSize);
351         out.updateNewSize();
352         outX = (int) (x + (compSize) - (compSize / 9.8));
353 
354         out.setBounds(outX, (int) y - out.getLinePointY(), out.getSize().width,
355             out.getSize().height);
356 
357         this.componentDim = new Dimension((int) outX + out.getBounds().width,
358             path.getBounds().height + 1);
359 
360       }
361     }
```

10.8 NandGate.java

```
1  package gates;
2 
3  import gui.DrawPanel;
4 
5  import java.awt.Dimension;
6  import java.awt.Point;
7  import java.awt.Shape;
8  import java.awt.geom.Ellipse2D;
```

```
9
10 import javax.swing.ButtonGroup;
11 import javax.swing.JCheckBoxMenuItem;
12 import javax.swing.JMenu;
13
14 /**
15  * NandGate.java
16  *
17  * @author Kandeeban Arumugam
18  */
19 public class NandGate extends Gate {
20   private int outX, outY;
21
22   /**
23    * Konstruktøren for NandGate-port.
24    *
25    * @param size - Størrelsen på porten.
26    * @param antallInp - Antall innganger til porten.
27    */
28   public NandGate(int size, int antallInp) {
29     this.compSize = size;
30     this.x = compSize * 2;
31     this.y = compSize;
32     this.antallInput = antallInp;
33     this.setLayout(null);
34
35     Shape circle = new Ellipse2D.Double(x + compSize + (compSize / 8), y
36         - (compSize / 4), (compSize / 2), (compSize / 2));
37
38     path.moveTo(x - compSize, y - compSize);
39     path.lineTo(x, y - compSize);
40     path.curveTo(x + ((compSize * 3) / 2), y - compSize, x
41         + ((compSize * 3) / 2), y + compSize, x, y + compSize);
42     path.lineTo(x - compSize, y + compSize);
43     path.closePath();
44     path.append(circle, false);
45
46     int inS = (int) (y * 2) / (antallInput);
47     int inSstart = inS / 2;
48
49     for (int i = 0; i < antallInput; i++) {
50       in[i] = new Input(compSize);
51       in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
52           .getSize().width, in[i].getSize().height);
53       inSstart += inS;
54       this.add(in[i]);
55     }
56     out = new Output(compSize);
57     outX = circle.getBounds().x + circle.getBounds().width;
58     outY = (int) y - out.getLinePointY();
59
60     out.setBounds(outX, outY, out.getSize().width, out.getSize().height);
61
62     this.add(out);
```

```
63      this.componentDim = new Dimension((int) outX + out.getBounds().width,
64          path.getBounds().height + 1);
65
66      makeInputPopupmenu(antallInput);
67    }
68
69
70    /**
71     * Denne metoden kan brukes til å gi mulighet for brukeren.
72     * slik at de kan selv bestemme antall innganger.
73     * @param antInput
74     */
75    public void makeInputPopupmenu(int antInput)
76    {
77      ButtonGroup group = new ButtonGroup();
78      JMenu inputSubMenu = new JMenu("Inputs");
79
80      //2
81      JCheckBoxMenuItem subMenuItem1 = new JCheckBoxMenuItem(antInput + "");
82      subMenuItem1.setSelected(true);
83      subMenuItem1.setActionCommand("inputs");
84      subMenuItem1.addActionListener(this.popListener);
85      inputSubMenu.add(subMenuItem1);
86      group.add(subMenuItem1);
87
88      //3
89      subMenuItem1 = new JCheckBoxMenuItem("3");
90      subMenuItem1.setSelected(true);
91      subMenuItem1.setActionCommand("inputs");
92      subMenuItem1.addActionListener(this.popListener);
93      inputSubMenu.add(subMenuItem1);
94      subMenuItem1.setEnabled(false);
95      group.add(subMenuItem1);
96
97      //4
98      subMenuItem1 = new JCheckBoxMenuItem("4");
99      subMenuItem1.setSelected(true);
100     subMenuItem1.setActionCommand("inputs");
101     subMenuItem1.addActionListener(this.popListener);
102     inputSubMenu.add(subMenuItem1);
103     subMenuItem1.setEnabled(false);
104     group.add(subMenuItem1);
105
106     popMenu.add(inputSubMenu);
107   }
108   /******* Metodene som brukes av XMLEncoder og XMLDecoder ******/
109
110   /**
111    * Tom konstrøktøren for NandGate-port.
112    *
113    * Denne metoden gjør ingenting, derfor må set-metodene brukes,
114    * til å angi nødvendige parametere.
115    *
116    * Deretter må create()-metoden kalles.
```

```
117     *
118     */
119    public NandGate() {
120    }
121
122    /**
123     * Denne metoden setteropp NandGate-porten, dersom nødvendige
124     * variablene er gitt ved å bruke set-metoder.
125     */
126    public void create() {
127      // TODO Auto-generated method stub
128      this.x = compSize * 2;
129      this.y = compSize;
130      this.setLayout(null);
131
132      Shape circle = new Ellipse2D.Double(x + compSize + (compSize / 8), y
133          - (compSize / 4), (compSize / 2), (compSize / 2));
134
135      path.moveTo(x - compSize, y - compSize);
136      path.lineTo(x, y - compSize);
137      path.curveTo(x + ((compSize * 3) / 2), y - compSize, x
138          + ((compSize * 3) / 2), y + compSize, x, y + compSize);
139      path.lineTo(x - compSize, y + compSize);
140      path.closePath();
141      path.append(circle, false);
142
143      for (int i = 0; i < antallInput; i++) {
144        in[i].create();
145        this.add(in[i]);
146      }
147
148      outX = circle.getBounds().x + circle.getBounds().width;
149      out.create();
150      this.add(out);
151      this.componentDim = new Dimension((int) outX + out.getBounds().width,
152          path.getBounds().height + 1);
153
154    }
155
156    /**
157     * Metoden tar imot utgangen for denne porten.
158     *
159     * @param out
160     */
161    public void setOut(Output out) {
162      this.out = out;
163    }
164
165    /**
166     * Metoden returnerer utgangen til denne porten.
167     *
168     * @return
169     */
170    public Output getOut() {
```

```
171       return this.out;
172     }
173
174     /**
175      * Metoden setter inngangene til denne porten.
176      *
177      * @param in
178      */
179     public void setIn(Input[] in) {
180       this.in = in;
181     }
182
183     /**
184      * Metoden returnerer inngangene til denne porten.
185      *
186      * @return
187      */
188     public Input[] getIn() {
189       return this.in;
190     }
191
192     /**
193      * Metoden kan brukes til å plassere porten.
194      *
195      * @param locPo
196      */
197     public void setLocPo(Point locPo) {
198       this.setLocation(locPo);
199     }
200
201     /**
202      * Metoden returnerer plasseringspunktet til porten.
203      *
204      * @return
205      */
206     public Point getLocPo() {
207       return this.getLocation();
208     }
209
210     /**
211      * Metoden kan brukes til å definere størrelsen.
212      *
213      * @param compSize
214      */
215     public void setCompSize(int compSize) {
216       this.compSize = compSize;
217     }
218
219     /**
220      * Metoden returnerer definerte størrelsen.
221      *
222      * @return
223      */
224     public int getCompSize() {
```

```
225       return this.compSize;
226     }
227
228     /**
229      * Metoden kan brukes til å sette antall innganger.
230      *
231      * @param antallInput
232      */
233     public void setAntallInput(int antallInput) {
234       this.antallInput = antallInput;
235     }
236
237     /**
238      * Returnerer antall innganger.
239      *
240      */
241     public int getAntallInput() {
242       return antallInput;
243     }
244
245     /**
246      * Returnerer størrelsen.
247      */
248     public Dimension getSize() {
249       return componentDim;
250     }
251
252     /********** END **************/
253
254     /**
255      * Metoden for å sette igang simulasjon.
256      *
257      * Metoden overrides Gate-metoden.
258      */
259     public synchronized void startSimulering() {
260
261       boolean ready = isReady();
262       if (ready) {
263
264         int sig = in[0].signal;
265         for (int i = 1; i < antallInput; i++) {
266           sig *= in[i].signal;
267         }
268
269         if (sig == 0) {
270           output_Signal = 1;
271         } else {
272           output_Signal = 0;
273         }
274         sendSignal();
275       }
276
277     }
278
```

```
279   /**
280    * Metoden sender videre utgangs-signalet.
281    */
282   public void sendSignal() {
283     DrawPanel parent = (DrawPanel) this.getParent();
284     final int sleep = parent.getClockSpeed();
285
286     WireElement wis = this.out.neste;
287     while (wis != null) {
288       wis.registerRots(rots, lastIncomingRot);
289       wis.setSignal(output_Signal);
290       wis.setShowCurent(true);
291       Thread t = Thread.currentThread();
292       try {
293         t.sleep(sleep);
294       } catch (InterruptedException e) {
295         // TODO Auto-generated catch block
296         e.printStackTrace();
297       }
298       wis = wis.neste;
299     }
300   }
301
302   /**
303    * Denne metoden brukes til å forstørre porten.
304    */
305   public void updateNewSize() {
306     this.x = compSize * 2;
307     this.y = compSize;
308     path.reset();
309
310     Shape circle = new Ellipse2D.Double(x + compSize + (compSize / 8), y
311         - (compSize / 4), (compSize / 2), (compSize / 2));
312
313     path.moveTo(x - compSize, y - compSize);
314     path.lineTo(x, y - compSize);
315     path.curveTo(x + ((compSize * 3) / 2), y - compSize, x
316         + ((compSize * 3) / 2), y + compSize, x, y + compSize);
317     path.lineTo(x - compSize, y + compSize);
318     path.closePath();
319     path.append(circle, false);
320
321     int inS = (int) (y * 2) / (antallInput);
322     int inSstart = inS / 2;
323
324     for (int i = 0; i < antallInput; i++) {
325       in[i].setCompSize(compSize);
326       in[i].updateNewSize();
327
328       in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
329           .getSize().width, in[i].getSize().height);
330       inSstart += inS;
331     }
332     out.setCompSize(compSize);
```

```
333         out.updateNewSize();
334
335         outX = circle.getBounds().x + circle.getBounds().width;
336         outY = (int) y - out.getLinePointY();
337
338         out.setBounds(outX, outY, out.getSize().width, out.getSize().height);
339
340         this.componentDim = new Dimension((int) outX + out.getBounds().width,
341             path.getBounds().height + 1);
342
343     }
344 }
```

10.9 NorGate.java

```
1  package gates;
2
3  import gui.DrawPanel;
4
5  import java.awt.Dimension;
6  import java.awt.Point;
7  import java.awt.Shape;
8  import java.awt.geom.Ellipse2D;
9
10 import javax.swing.ButtonGroup;
11 import javax.swing.JCheckBoxMenuItem;
12 import javax.swing.JMenu;
13
14 /**
15  * NorGate.java
16  *
17  * @author Kandeeban Arumugam
18  */
19 public class NorGate extends Gate {
20   private int outX;
21
22   /**
23    * Konstruktøren for NorGate-port.
24    *
25    * @param size - Størrelsen på porten.
26    * @param antallInp - Antall innganger til porten.
27    */
28   public NorGate(int size, int antallInp) {
29     this.compSize = size;
30     this.x = size * 2;
31     this.y = size;
32     this.antallInput = antallInp;
33     this.setLayout(null);
34
35     Shape circle = new Ellipse2D.Double(x + (compSize) - (compSize / 9.8),
36         y - (compSize / 4), (compSize / 2), (compSize / 2));
```

```
37      path.moveTo(x - compSize - (compSize / 3), y + compSize);
38      path.quadTo(x - (compSize / 2.4), y, x - compSize - (compSize / 3), y
39          - compSize);
40      path.lineTo(x - (compSize / 3), y - compSize);
41      path.curveTo(x + (compSize / 3), y - (compSize / 1.2),
42          x + compSize / 2, y - (compSize / 1.5), x + (compSize)
43              - (compSize / 9.8), y);
44      path.curveTo(x + compSize / 2, y + (compSize / 1.5),
45          x + (compSize / 3), y + (compSize / 1.2), x - (compSize / 3), y
46              + compSize);
47      path.closePath();
48      path.append(circle, false);
49
50      int inS = (int) (y * 2) / (antallInput);
51      int inSstart = inS / 2;
52
53      for (int i = 0; i < antallInput; i++) {
54        in[i] = new Input(compSize);
55        in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
56            .getSize().width, in[i].getSize().height);
57        inSstart += inS;
58        this.add(in[i]);
59      }
60      out = new Output(compSize);
61      outX = circle.getBounds().x + circle.getBounds().width;
62
63      out.setBounds(outX, (int) y - out.getLinePointY(), out.getSize().width,
64          out.getSize().height);
65
66      this.add(out);
67      this.componentDim = new Dimension((int) outX + out.getBounds().width,
68          path.getBounds().height + 1);
69
70      makeInputPopupmenu(antallInput);
71    }
72
73    /**
74     * Denne metoden kan brukes til å gi mulighet for brukeren.
75     * slik at de kan selv bestemme antall innganger.
76     * @param antInput
77     */
78    public void makeInputPopupmenu(int antInput)
79    {
80      ButtonGroup group = new ButtonGroup();
81      JMenu inputSubMenu = new JMenu("Inputs");
82
83      //2
84      JCheckBoxMenuItem subMenuItem1 = new JCheckBoxMenuItem(antInput + "");
85      subMenuItem1.setSelected(true);
86      subMenuItem1.setActionCommand("inputs");
87      subMenuItem1.addActionListener(this.popListener);
88      inputSubMenu.add(subMenuItem1);
89      group.add(subMenuItem1);
90
```

```
91      //3
92      subMenuItem1 = new JCheckBoxMenuItem("3");
93      subMenuItem1.setSelected(true);
94      subMenuItem1.setActionCommand("inputs");
95      subMenuItem1.addActionListener(this.popListener);
96      inputSubMenu.add(subMenuItem1);
97      subMenuItem1.setEnabled(false);
98      group.add(subMenuItem1);
99
100     //4
101     subMenuItem1 = new JCheckBoxMenuItem("4");
102     subMenuItem1.setSelected(true);
103     subMenuItem1.setActionCommand("inputs");
104     subMenuItem1.addActionListener(this.popListener);
105     inputSubMenu.add(subMenuItem1);
106     subMenuItem1.setEnabled(false);
107     group.add(subMenuItem1);
108
109     popMenu.add(inputSubMenu);
110   }
111
112   /**** Metodene som brukes av XMLEncoder og XMLDecoder ********/
113   /**
114    * Tom konstrøktøren for NorGate-port.
115    *
116    * Denne metoden gjør ingenting, derfor må set-metodene brukes,
117    * til å angi nødvendige parametere.
118    *
119    * Deretter må create()-metoden kalles.
120    *
121    */
122   public NorGate() {
123   }
124
125   /**
126    * Denne metoden setteropp NorGate-porten, dersom nødvendige
127    * variablene er gitt ved å bruke set-metoder.
128    */
129   public void create() {
130     // TODO Auto-generated method stub
131     this.x = compSize * 2;
132     this.y = compSize;
133     this.setLayout(null);
134
135     Shape circle = new Ellipse2D.Double(x + (compSize) - (compSize / 9.8),
136         y - (compSize / 4), (compSize / 2), (compSize / 2));
137     path.moveTo(x - compSize - (compSize / 3), y + compSize);
138     path.quadTo(x - (compSize / 2.4), y, x - compSize - (compSize / 3), y
139         - compSize);
140     path.lineTo(x - (compSize / 3), y - compSize);
141     path.curveTo(x + (compSize / 3), y - (compSize / 1.2),
142         x + compSize / 2, y - (compSize / 1.5), x + (compSize)
143             - (compSize / 9.8), y);
144     path.curveTo(x + compSize / 2, y + (compSize / 1.5),
```

```
145             x + (compSize / 3), y + (compSize / 1.2), x - (compSize / 3), y
146                 + compSize);
147         path.closePath();
148         path.append(circle, false);
149 
150         for (int i = 0; i < antallInput; i++) {
151           in[i].create();
152           this.add(in[i]);
153         }
154 
155         outX = circle.getBounds().x + circle.getBounds().width;
156         out.create();
157         this.add(out);
158         this.componentDim = new Dimension((int) outX + out.getBounds().width,
159             path.getBounds().height + 1);
160 
161       }
162 
163       /**
164        * Metoden tar imot utgangen for denne porten.
165        *
166        * @param out
167        */
168       public void setOut(Output out) {
169         this.out = out;
170       }
171 
172       /**
173        * Metoden returnerer utgangen til denne porten.
174        *
175        * @return
176        */
177       public Output getOut() {
178         return this.out;
179       }
180 
181       /**
182        * Metoden setter inngangene til denne porten.
183        *
184        * @param in
185        */
186       public void setIn(Input[] in) {
187         this.in = in;
188       }
189 
190       /**
191        * Metoden returnerer inngangene til denne porten.
192        *
193        * @return
194        */
195       public Input[] getIn() {
196         return this.in;
197       }
198 
```

```
199     /**
200      * Metoden kan brukes til å plassere porten.
201      *
202      * @param locPo
203      */
204     public void setLocPo(Point locPo) {
205       this.setLocation(locPo);
206     }
207 
208     /**
209      * Metoden returnerer plasseringspunktet til porten.
210      *
211      * @return
212      */
213     public Point getLocPo() {
214       return this.getLocation();
215     }
216 
217     /**
218      * Metoden kan brukes til å definere størrelsen.
219      *
220      * @param compSize
221      */
222     public void setCompSize(int compSize) {
223       this.compSize = compSize;
224     }
225 
226     /**
227      * Metoden returnerer definerte størrelsen.
228      *
229      * @return
230      */
231     public int getCompSize() {
232       return this.compSize;
233     }
234 
235     /**
236      * Metoden kan brukes til å sette antall innganger.
237      *
238      * @param antallInput
239      */
240     public void setAntallInput(int antallInput) {
241       this.antallInput = antallInput;
242     }
243 
244     /**
245      * Returnerer antall innganger.
246      *
247      */
248     public int getAntallInput() {
249       return antallInput;
250     }
251 
252     /**
```

```
253     * Returnerer størrelsen.
254     */
255    public Dimension getSize() {
256      return componentDim;
257    }
258
259    /********** END ***************/
260    /**
261     * Metoden for å sette igang simulasjon.
262     *
263     * Metoden overrides Gate-metoden.
264     */
265    public synchronized void startSimulering() {
266
267      boolean ready = isReady();
268      if (ready) {
269
270        int sig = in[0].signal;
271        for (int i = 1; i < antallInput; i++) {
272          sig |= in[i].signal;
273        }
274
275        if (sig == 0) {
276          output_Signal = 1;
277        } else {
278          output_Signal = 0;
279        }
280        sendSignal();
281
282      }
283    }
284
285    /**
286     * Metoden sender videre utgangs-signalet.
287     */
288    public void sendSignal() {
289      DrawPanel parent = (DrawPanel) this.getParent();
290      final int sleep = parent.getClockSpeed();
291
292      WireElement wis = this.out.neste;
293      while (wis != null) {
294        wis.registerRots(rots, lastIncomingRot);
295        wis.setSignal(output_Signal);
296        wis.setShowCurent(true);
297        Thread t = Thread.currentThread();
298        try {
299          t.sleep(sleep);
300        } catch (InterruptedException e) {
301          // TODO Auto-generated catch block
302          e.printStackTrace();
303        }
304        wis = wis.neste;
305      }
306    }
```

```
307
308     /**
309      * Denne metoden brukes til å forstørre porten.
310      */
311     public void updateNewSize() {
312       this.x = compSize * 2;
313       this.y = compSize;
314       path.reset();
315
316       Shape circle = new Ellipse2D.Double(x + (compSize) - (compSize / 9.8),
317           y - (compSize / 4), (compSize / 2), (compSize / 2));
318       path.moveTo(x - compSize - (compSize / 3), y + compSize);
319       path.quadTo(x - (compSize / 2.4), y, x - compSize - (compSize / 3), y
320           - compSize);
321       path.lineTo(x - (compSize / 3), y - compSize);
322       path.curveTo(x + (compSize / 3), y - (compSize / 1.2),
323           x + compSize / 2, y - (compSize / 1.5), x + (compSize)
324               - (compSize / 9.8), y);
325       path.curveTo(x + compSize / 2, y + (compSize / 1.5),
326           x + (compSize / 3), y + (compSize / 1.2), x - (compSize / 3), y
327               + compSize);
328       path.closePath();
329       path.append(circle, false);
330
331       int inS = (int) (y * 2) / (antallInput);
332       int inSstart = inS / 2;
333
334       for (int i = 0; i < antallInput; i++) {
335         in[i].setCompSize(compSize);
336         in[i].updateNewSize();
337
338         in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
339             .getSize().width, in[i].getSize().height);
340         inSstart += inS;
341       }
342       out.setCompSize(compSize);
343       out.updateNewSize();
344
345       outX = circle.getBounds().x + circle.getBounds().width;
346
347       out.setBounds(outX, (int) y - out.getLinePointY(), out.getSize().width,
348           out.getSize().height);
349
350       this.componentDim = new Dimension((int) outX + out.getBounds().width,
351           path.getBounds().height + 1);
352
353     }
354
355 }
```

10.10 NotGate.java

```
1  package gates;
2
3  import gui.DrawPanel;
4
5  import java.awt.Dimension;
6  import java.awt.Point;
7  import java.awt.Shape;
8  import java.awt.geom.Ellipse2D;
9
10 /**
11  * NotGate.java
12  *
13  * @author Kandeeban Arumugam
14  */
15 public class NotGate extends Gate {
16
17   /**
18    * Konstruktøren for NotGate-port.
19    *
20    * @param size - Størrelsen på porten.
21    * @param antallInp - Antall innganger til porten.
22    */
23   public NotGate(int size, int antallInp) {
24     this.x = size * 2;
25     this.y = size;
26     this.compSize = size;
27     this.antallInput = antallInp;
28     this.setLayout(null);
29     Shape circle = new Ellipse2D.Double(x + (size), y - (size / 4),
30         (size / 2), (size / 2));
31
32     path.moveTo((x) - size, (y));
33     path.lineTo((x - size), (y) - size);
34     path.lineTo((x + (size)), (y));
35     path.lineTo((x - size), size + (y));
36     path.closePath();
37     path.append(circle, false);
38     int inS = (int) (y * 2) / (antallInput);
39     int inSstart = inS / 2;
40
41     for (int i = 0; i < antallInput; i++) {
42       in[i] = new Input(size);
43       in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
44           .getSize().width, in[i].getSize().height);
45       inSstart += inS;
46       this.add(in[i]);
47     }
48     out = new Output(size);
49     int outX = circle.getBounds().x + circle.getBounds().width;
50
51     out.setBounds(outX, (int) y - out.getLinePointY(), out.getSize().width,
52         out.getSize().height);
53
```

```
54      this.add(out);
55
56      this.componentDim = new Dimension((int) path.getBounds().getWidth()
57          + in[0].getSize().width + out.getBounds().width - 1, path
58          .getBounds().height + 1);
59    }
60
61    /*******Metodene som brukes av XMLEncoder og XMLDecoder*******/
62    /**
63     * Tom konstrøktøren for NotGate-port.
64     *
65     * Denne metoden gjør ingenting, derfor må set-metodene brukes,
66     * til å angi nødvendige parametere.
67     *
68     * Deretter må create()-metoden kalles.
69     *
70     */
71    public NotGate() {
72    }
73
74    /**
75     * Denne metoden setteropp NotGate-porten, dersom nødvendige
76     * variablene er gitt ved å bruke set-metoder.
77     */
78    public void create() {
79      this.setLayout(null);
80      this.x = compSize * 2;
81      this.y = compSize;
82      Shape circle = new Ellipse2D.Double(x + (compSize), y - (compSize / 4),
83          (compSize / 2), (compSize / 2));
84
85      path.moveTo((x) - compSize, (y));
86      path.lineTo((x - compSize), (y) - compSize);
87      path.lineTo((x + (compSize)), (y));
88      path.lineTo((x - compSize), compSize + (y));
89      path.closePath();
90      path.append(circle, false);
91
92      for (int i = 0; i < antallInput; i++) {
93        in[i].create();
94        this.add(in[i]);
95      }
96
97      out.create();
98      this.add(out);
99
100     this.componentDim = new Dimension((int) path.getBounds().getWidth()
101         + in[0].getSize().width + out.getBounds().width - 1, path
102         .getBounds().height + 1);
103   }
104
105   /**
106    * Metoden tar imot utgangen for denne porten.
107    *
```

```
108     * @param out
109     */
110    public void setOut(Output out)
111    {
112      this.out = out;
113    }
114
115    /**
116     * Metoden returnerer utgangen til denne porten.
117     *
118     * @return
119     */
120    public Output getOut()
121    {
122      return this.out;
123    }
124
125    /**
126     * Metoden setter inngangene til denne porten.
127     *
128     * @param in
129     */
130    public void setIn(Input[] in)
131    {
132      this.in = in;
133    }
134
135    /**
136     * Metoden returnerer inngangene til denne porten.
137     *
138     * @return
139     */
140    public Input[] getIn()
141    {
142      return this.in;
143    }
144
145    /**
146     * Metoden kan brukes til å plassere porten.
147     *
148     * @param locPo
149     */
150    public void setLocPo(Point locPo)
151    {
152      this.setLocation(locPo);
153    }
154
155    /**
156     * Metoden returnerer plasseringspunktet til porten.
157     *
158     * @return
159     */
160    public Point getLocPo()
161    {
```

```
162       return this.getLocation();
163     }
164
165     /**
166      * Metoden kan brukes til å definere størrelsen.
167      *
168      * @param compSize
169      */
170     public void setCompSize(int compSize) {
171       this.compSize = compSize;
172     }
173
174     /**
175      * Metoden returnerer definerte størrelsen.
176      *
177      * @return
178      */
179     public int getCompSize() {
180       return this.compSize;
181     }
182
183     /**
184      * Metoden kan brukes til å sette antall innganger.
185      *
186      * @param antallInput
187      */
188     public void setAntallInput(int antallInput) {
189       this.antallInput = antallInput;
190     }
191
192     /**
193      * Returnerer antall innganger.
194      *
195      */
196     public int getAntallInput() {
197       return antallInput;
198     }
199
200     /**
201      * Returnerer størrelsen.
202      */
203     public Dimension getSize() {
204       return componentDim;
205     }
206 /**************END*****************/
207     /**
208      * Metoden for å sette igang simulasjon.
209      *
210      * Metoden overrides Gate-metoden.
211      */
212     public synchronized void startSimulering() {
213
214       int sig = in[0].signal;
215
```

```
216       if (sig == 0) {
217         output_Signal = 1;
218       } else {
219         output_Signal = 0;
220       }
221       sendSignal();
222     }
223
224     /**
225      * Metoden sender videre utgangs-signalet.
226      */
227     public void sendSignal() {
228       DrawPanel parent = (DrawPanel) this.getParent();
229       final int sleep = parent.getClockSpeed();
230
231       WireElement wis = this.out.neste;
232       while (wis != null) {
233         wis.registerRots(rots, lastIncomingRot);
234         wis.setSignal(output_Signal);
235         wis.setShowCurent(true);
236         Thread t = Thread.currentThread();
237         try {
238           t.sleep(sleep);
239         } catch (InterruptedException e) {
240           e.printStackTrace();
241         }
242         wis = wis.neste;
243       }
244     }
245
246     /**
247      * Denne metoden brukes til å forstørre porten.
248      */
249     public void updateNewSize() {
250       this.x = compSize * 2;
251       this.y = compSize;
252       path.reset();
253
254       Shape circle = new Ellipse2D.Double(x + (compSize), y - (compSize / 4),
255           (compSize / 2), (compSize / 2));
256
257       path.moveTo((x) - compSize, (y));
258       path.lineTo((x - compSize), (y) - compSize);
259       path.lineTo((x + (compSize)), (y));
260       path.lineTo((x - compSize), compSize + (y));
261       path.closePath();
262       path.append(circle, false);
263       int inS = (int) (y * 2) / (antallInput);
264       int inSstart = inS / 2;
265
266       for (int i = 0; i < antallInput; i++) {
267         in[i].setCompSize(compSize);
268         in[i].updateNewSize();
269
```

```
270         in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
271             .getSize().width, in[i].getSize().height);
272         inSstart += inS;
273       }
274       out.setCompSize(compSize);
275       out.updateNewSize();
276
277       int outX = circle.getBounds().x + circle.getBounds().width;
278
279       out.setBounds(outX, (int) y - out.getLinePointY(), out.getSize().width,
280           out.getSize().height);
281
282       this.componentDim = new Dimension((int) path.getBounds().getWidth()
283           + in[0].getSize().width + out.getBounds().width - 1, path
284           .getBounds().height + 1);
285
286     }
287
288 }
```

10.11 OrGate.java

```
1  package gates;
2
3  import gui.DrawPanel;
4
5  import java.awt.Dimension;
6  import java.awt.Point;
7
8  import javax.swing.ButtonGroup;
9  import javax.swing.JCheckBoxMenuItem;
10 import javax.swing.JMenu;
11
12 /**
13  * OrGate.java
14  *
15  * @author Kandeeban Arumugam
16  */
17 public class OrGate extends Gate {
18   private int outX;
19
20   /**
21    * Konstruktøren for OrGate-port.
22    *
23    * @param size - Størrelsen på porten.
24    * @param antallInp - Antall innganger til porten.
25    */
26   public OrGate(int size, int antallInp) {
27
28     this.compSize = size;
29     this.x = compSize * 2;
```

```
30      this.y = compSize;
31      this.antallInput = antallInp;
32      this.setLayout(null);
33
34      path.moveTo(x - compSize - (compSize / 3), y + compSize);
35      path.quadTo(x - (compSize / 2.4), y, x - compSize - (compSize / 3), y
36          - compSize);
37      path.lineTo(x - (compSize / 3), y - compSize);
38      path.curveTo(x + (compSize / 3), y - (compSize / 1.2),
39          x + compSize / 2, y - (compSize / 1.5), x + (compSize)
40              - (compSize / 9.8), y);
41      path.curveTo(x + compSize / 2, y + (compSize / 1.5),
42          x + (compSize / 3), y + (compSize / 1.2), x - (compSize / 3), y
43              + compSize);
44      path.closePath();
45
46      int inS = (int) (y * 2) / (antallInput);
47      int inSstart = inS / 2;
48
49      for (int i = 0; i < antallInput; i++) {
50        in[i] = new Input(compSize);
51        in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
52            .getSize().width, in[i].getSize().height);
53        inSstart += inS;
54        this.add(in[i]);
55      }
56      out = new Output(compSize);
57      outX = (int) (x + (compSize) - (compSize / 9.8));
58
59      out.setBounds(outX, (int) y - out.getLinePointY(), out.getSize().width,
60          out.getSize().height);
61
62      this.add(out);
63      this.componentDim = new Dimension((int) outX + out.getBounds().width,
64          path.getBounds().height + 1);
65
66      makeInputPopupmenu(antallInput);
67    }
68
69    /**
70     * Denne metoden kan brukes til å gi mulighet for brukeren.
71     * slik at de kan selv bestemme antall innganger.
72     * @param antInput
73     */
74    public void makeInputPopupmenu(int antInput)
75    {
76      ButtonGroup group = new ButtonGroup();
77      JMenu inputSubMenu = new JMenu("Inputs");
78
79      //2
80      JCheckBoxMenuItem subMenuItem1 = new JCheckBoxMenuItem(antInput + "");
81      subMenuItem1.setSelected(true);
82      subMenuItem1.setActionCommand("inputs");
83      subMenuItem1.addActionListener(this.popListener);
```

```
84        inputSubMenu.add(subMenuItem1);
85        group.add(subMenuItem1);
86
87        //3
88        subMenuItem1 = new JCheckBoxMenuItem("3");
89        subMenuItem1.setSelected(true);
90        subMenuItem1.setActionCommand("inputs");
91        subMenuItem1.addActionListener(this.popListener);
92        inputSubMenu.add(subMenuItem1);
93        subMenuItem1.setEnabled(false);
94        group.add(subMenuItem1);
95
96        //4
97        subMenuItem1 = new JCheckBoxMenuItem("4");
98        subMenuItem1.setSelected(true);
99        subMenuItem1.setActionCommand("inputs");
100       subMenuItem1.addActionListener(this.popListener);
101       inputSubMenu.add(subMenuItem1);
102       subMenuItem1.setEnabled(false);
103       group.add(subMenuItem1);
104
105       popMenu.add(inputSubMenu);
106   }
107
108   /****** Metodene som brukes av XMLEncoder og XMLDecoder *******/
109   /**
110    * Tom konstrøktøren for OrGate-port.
111    *
112    * Denne metoden gjør ingenting, derfor må set-metodene brukes,
113    * til å angi nødvendige parametere.
114    *
115    * Deretter må create()-metoden kalles.
116    *
117    */
118   public OrGate() {
119   }
120
121   /**
122    * Denne metoden setteropp OrGate-porten, dersom nødvendige
123    * variablene er gitt ved å bruke set-metoder.
124    */
125   public void create() {
126       // TODO Auto-generated method stub
127       this.x = compSize * 2;
128       this.y = compSize;
129       this.setLayout(null);
130
131       path.moveTo(x - compSize - (compSize / 3), y + compSize);
132       path.quadTo(x - (compSize / 2.4), y, x - compSize - (compSize / 3), y
133           - compSize);
134       path.lineTo(x - (compSize / 3), y - compSize);
135       path.curveTo(x + (compSize / 3), y - (compSize / 1.2),
136           x + compSize / 2, y - (compSize / 1.5), x + (compSize)
137               - (compSize / 9.8), y);
```

```
138       path.curveTo(x + compSize / 2, y + (compSize / 1.5),
139           x + (compSize / 3), y + (compSize / 1.2), x - (compSize / 3), y
140               + compSize);
141       path.closePath();
142
143       for (int i = 0; i < antallInput; i++) {
144         in[i].create();
145         this.add(in[i]);
146       }
147       outX = (int) (x + (compSize) - (compSize / 9.8));
148       out.create();
149       this.add(out);
150       this.componentDim = new Dimension((int) outX + out.getBounds().width,
151           path.getBounds().height + 1);
152
153     }
154
155     /**
156      * Metoden tar imot utgangen for denne porten.
157      *
158      * @param out
159      */
160     public void setOut(Output out) {
161       this.out = out;
162     }
163
164     /**
165      * Metoden returnerer utgangen til denne porten.
166      *
167      * @return
168      */
169     public Output getOut() {
170       return this.out;
171     }
172
173     /**
174      * Metoden setter inngangene til denne porten.
175      *
176      * @param in
177      */
178     public void setIn(Input[] in) {
179       this.in = in;
180     }
181
182     /**
183      * Metoden returnerer inngangene til denne porten.
184      *
185      * @return
186      */
187     public Input[] getIn() {
188       return this.in;
189     }
190
191     /**
```

```
192     * Metoden kan brukes til å plassere porten.
193     *
194     * @param locPo
195     */
196    public void setLocPo(Point locPo) {
197      this.setLocation(locPo);
198    }
199
200    /**
201     * Metoden returnerer plasseringspunktet til porten.
202     *
203     * @return
204     */
205    public Point getLocPo() {
206      return this.getLocation();
207    }
208
209    /**
210     * Metoden kan brukes til å definere størrelsen.
211     *
212     * @param compSize
213     */
214    public void setCompSize(int compSize) {
215      this.compSize = compSize;
216    }
217
218    /**
219     * Metoden returnerer definerte størrelsen.
220     *
221     * @return
222     */
223    public int getCompSize() {
224      return this.compSize;
225    }
226
227    /**
228     * Metoden kan brukes til å sette antall innganger.
229     *
230     * @param antallInput
231     */
232    public void setAntallInput(int antallInput) {
233      this.antallInput = antallInput;
234    }
235
236    /**
237     * Returnerer antall innganger.
238     *
239     */
240    public int getAntallInput() {
241      return antallInput;
242    }
243
244    /**
245     * Returnerer størrelsen.
```

```
246     */
247    public Dimension getSize() {
248      return componentDim;
249    }
250
251    /************** END ******************/
252    /**
253     * Metoden for å sette igang simulasjon.
254     *
255     * Metoden overrides Gate-metoden.
256     */
257    public synchronized void startSimulering() {
258
259      boolean ready = isReady();
260      if (ready) {
261        int sig = in[0].signal;
262        for (int i = 1; i < antallInput; i++) {
263          sig |= in[i].signal;
264        }
265        output_Signal = sig;
266
267        sendSignal();
268      }
269    }
270
271    /**
272     * Metoden sender videre utgangs-signalet.
273     */
274    public void sendSignal() {
275      DrawPanel parent = (DrawPanel) this.getParent();
276      final int sleep = parent.getClockSpeed();
277
278      WireElement wis = this.out.neste;
279      while (wis != null) {
280        wis.registerRots(rots, lastIncomingRot);
281        wis.setSignal(output_Signal);
282        wis.setShowCurent(true);
283        Thread t = Thread.currentThread();
284        try {
285          t.sleep(sleep);
286        } catch (InterruptedException e) {
287          // TODO Auto-generated catch block
288          e.printStackTrace();
289        }
290        wis = wis.neste;
291      }
292    }
293
294    /**
295     * Denne metoden brukes til å forstørre porten.
296     */
297    public void updateNewSize() {
298      this.x = compSize * 2;
299      this.y = compSize;
```

```
300         path.reset();
301
302         path.moveTo(x - compSize - (compSize / 3), y + compSize);
303         path.quadTo(x - (compSize / 2.4), y, x - compSize - (compSize / 3), y
304             - compSize);
305         path.lineTo(x - (compSize / 3), y - compSize);
306         path.curveTo(x + (compSize / 3), y - (compSize / 1.2),
307             x + compSize / 2, y - (compSize / 1.5), x + (compSize)
308                 - (compSize / 9.8), y);
309         path.curveTo(x + compSize / 2, y + (compSize / 1.5),
310             x + (compSize / 3), y + (compSize / 1.2), x - (compSize / 3), y
311                 + compSize);
312         path.closePath();
313
314         int inS = (int) (y * 2) / (antallInput);
315         int inSstart = inS / 2;
316
317         for (int i = 0; i < antallInput; i++) {
318           in[i].setCompSize(compSize);
319           in[i].updateNewSize();
320
321           in[i].setBounds(0, inSstart - in[i].getLinePointY(), in[i]
322               .getSize().width, in[i].getSize().height);
323           inSstart += inS;
324         }
325         out.setCompSize(compSize);
326         out.updateNewSize();
327         outX = (int) (x + (compSize) - (compSize / 9.8));
328
329         out.setBounds(outX, (int) y - out.getLinePointY(), out.getSize().width,
330             out.getSize().height);
331
332         this.componentDim = new Dimension((int) outX + out.getBounds().width,
333             path.getBounds().height + 1);
334
335       }
336
337     }
```

10.12 WireElement.java

```
1   package gates;
2
3
4   import java.awt.Color;
5   import java.util.HashMap;
6
7   import javax.swing.ToolTipManager;
8
9   /**
10   * WireElement.java
```

```
11  *
12  * @author Kandeeban Arumugam
13  */
14 public abstract class WireElement extends Elements{
15
16   protected WireElement neste, forrige;
17   protected int signal;
18   protected int compSize;
19   protected boolean signalRegistered;
20   protected int registeredElement;
21   protected int antallRegSignal;
22   protected boolean showCurent;
23   protected Color color;
24   HashMap<Integer, Integer> rotAndForeldre;
25
26   public WireElement(){
27     rotAndForeldre = new HashMap<Integer, Integer>();
28     signalRegistered = false;
29     showCurent = false;
30     color = Color.BLACK;
31
32     registeredElement = -1;
33     antallRegSignal = 0;
34
35     signal = 0;
36     neste = null;
37     forrige = null;
38     ToolTipManager.sharedInstance().setInitialDelay(0);
39   }
40
41   public void setSignal(int value)
42   {
43     setSignalRegistered(true);
44     antallRegSignal++;
45     if(this.getParent() instanceof Gate)
46     {
47       signal = value;
48       Gate ga = (Gate) this.getParent();
49       //ga.registerElement(this.getRegisteredElement());
50       ga.startSimulering();
51     }
52     else if (this.getParent() instanceof Led)
53     {
54       signal = value;
55       Led le = (Led) this.getParent();
56       le.signal = value;
57       le.repaint();
58 //       System.out.println("Led " + Thread.currentThread().getName());
59     }
60     else if(this instanceof ExtendsPoint)
61     {
62       signal = value;
63       ExtendsPoint ex = (ExtendsPoint)this;
64       ex.startSimulering();
```

```
65      }
66      else
67      {
68        signal = value;
69      }
70
71    }
72
73    public int getRegAntallSig()
74    {
75      return this.antallRegSignal;
76    }
77
78    public int getRegisteredSignal()
79    {
80      return signal;
81    }
82
83    public boolean isSignalRegistered()
84    {
85      return signalRegistered;
86    }
87
88    public void setSignalRegistered(boolean value)
89    {
90      this.signalRegistered = value;
91    }
92
93    public void registerRots(HashMap rot, int lastIncomingRot)
94    {
95      if(rot != null)
96      {
97        rotAndForeldre.putAll(rot);
98      }
99      if(this.getParent() instanceof Gate)
100     {
101       Gate ga = (Gate) this.getParent();
102       ga.lastIncomingRot = lastIncomingRot;
103     }
104
105     this.registeredElement = lastIncomingRot;
106   }
107
108   //Denne metoden kun benyttes av InteravtiveInput
109   //Det vi si, de som er koblet direkte til porten
110   public void registerElement(int id)
111   {
112     this.registeredElement = id;
113     if(rotAndForeldre.containsKey(id))
114     {
115       int value = rotAndForeldre.get(id);
116       rotAndForeldre.put(id, ++value);
117     }
118     else
```

```
119       {
120         rotAndForeldre.put(id, 1);
121       }
122       if(this.getParent() instanceof Gate)
123       {
124         Gate ga = (Gate) this.getParent();
125         ga.lastIncomingRot = id;
126       }
127     }
128
129     public int getRegisteredElement()
130     {
131       return this.registeredElement;
132     }
133
134     public WireElement getNeste()
135     {
136       return neste;
137     }
138
139     public WireElement getForrige()
140     {
141       return forrige;
142     }
143
144     public void setForrige(WireElement forrige)
145     {
146       this.forrige = forrige;
147     }
148
149     public void setNeste(WireElement neste)
150     {
151       this.neste = neste;
152     }
153
154     public void setShowCurent(boolean show)
155     {
156       this.showCurent = show;
157       if(signal == 0)
158       {
159         color = Color.BLACK;
160       }
161       else
162       {
163         color = Color.RED;
164       }
165       this.repaint();
166     }
167
168     public void reset()
169     {
170       color = Color.BLACK;
171       this.repaint();
172     }
```

```
173 }
```

10.13 Input.java

```
1  package gates;
2
3  import gui.DrawPanel;
4  import java.awt.Color;
5  import java.awt.Component;
6  import java.awt.Dimension;
7  import java.awt.Graphics;
8  import java.awt.Graphics2D;
9  import java.awt.Point;
10 import java.awt.event.MouseEvent;
11 import java.awt.event.MouseListener;
12 import java.awt.event.MouseMotionListener;
13 import java.awt.geom.GeneralPath;
14 import javax.swing.SwingUtilities;
15
16 /**
17  * Input.java
18  *
19  * @author Kandeeban Arumugam
20  *
21  */
22 public class Input extends WireElement implements MouseListener,
23                         MouseMotionListener{
24   private GeneralPath path;
25   private float x, y;
26   private int thisHeight;
27
28   private Point connecting_point;
29   boolean connected, tipVisable;
30   private Dimension compDim;
31   private int compSize;
32   private int updateP;
33
34   /**
35    * Konstruktøren tar imot størrelsen for inngang
36    *
37    * @param size
38    */
39   public Input(int size)
40   {
41     path = new GeneralPath();
42     this.compSize = size;
43     this.thisHeight = 6;
44     this.x = 0;
45     this.y = 3;
46     this.updateP = 0;
47
```

```
48      forrige = null;
49      neste = null;
50
51      connected = false;
52      tipVisable = false;
53
54      connecting_point = new Point();
55      connecting_point.setLocation(x, y);
56
57      path.moveTo(x, (y));
58      path.lineTo((x) + compSize, (y));
59      path.closePath();
60      this.addMouseListener(this);
61      this.addMouseMotionListener(this);
62      this.compDim = new Dimension((int) path.getBounds().getWidth()
63      + 1, path.getBounds().height + 1 + thisHeight);
64    }
65
66    public Input(){
67      path = new GeneralPath();
68      connecting_point = new Point();
69    }
70
71    public void create()
72    {
73      this.thisHeight = 6;
74      this.x = 0;
75      this.y = 3;
76
77      connecting_point.setLocation(x, y);
78
79      path.moveTo(x, (y));
80      path.lineTo((x) + compSize, (y));
81      path.closePath();
82
83      this.compDim = new Dimension((int) path.getBounds().getWidth()
84      + 1, path.getBounds().height + 1 + thisHeight);
85
86    }
87
88    public void updateNewSize()
89    {
90      path.reset();
91
92      path.moveTo(x, (y));
93      path.lineTo((x) + compSize, (y));
94      path.closePath();
95
96      this.compDim = new Dimension((int) path.getBounds().getWidth()
97      + 1, path.getBounds().height + 1 + thisHeight);
98    }
99
100   public void setConnected(boolean connected)
101   {
```

```
102       this.connected = connected;
103     }
104
105     public boolean getConnected()
106     {
107       return this.connected;
108     }
109     public void setCompDim(Dimension compDim)
110     {
111       this.compDim = compDim;
112     }
113
114     public Dimension getCompDim()
115     {
116       return this.compDim;
117     }
118     public void setCompSize(int compSize)
119     {
120       this.compSize = compSize;
121     }
122
123     public int getCompSize()
124     {
125       return this.compSize;
126     }
127
128     public void setConnecting_point(Point connecting_point)
129     {
130       this.connecting_point = connecting_point;
131     }
132
133     public Point getConnecting_point()
134     {
135       return this.connecting_point;
136     }
137
138      protected Point getConnectionPoint(Component parent)
139      {
140        return SwingUtilities.convertPoint(this,connecting_point, parent);
141      }
142
143     public void paintComponent(Graphics g) {
144       Graphics2D g2 = (Graphics2D) g;
145       g2.setPaint(Color.YELLOW);
146 //      g2.setPaint(Color.WHITE);
147       g2.fillRect(0, 0, getWidth(), getHeight());
148       g2.fill(path);
149       g2.setPaint(Color.BLACK);
150       g2.draw(path);
151       if(compDim != null)
152       {
153         this.setSize(compDim);
154       }
155     }
```

```
156
157   protected int getLinePointY()
158   {
159     return this.thisHeight/2;
160   }
161
162   public Dimension getSize() {
163     return compDim;
164   }
165
166   public int getUpdatePNr()
167   {
168     return updateP;
169   }
170
171   public void setUpdatePNr(int updateP)
172   {
173     this.updateP = updateP;
174   }
175
176   public void mouseClicked(MouseEvent e) {
177     // TODO Auto-generated method stub
178     DrawPanel parent = (DrawPanel) this.getParent().getParent();
179     Point containerPoint;
180     if(parent.isWireModus())
181     {
182     if (!connected) {
183
184       if (!parent.checkConnectedFirst()) {
185         containerPoint = SwingUtilities.convertPoint(this,
186             connecting_point, parent);
187         parent.setFirstPoint(containerPoint, this);
188         connected = true;
189         updateP = 1;
190       } else {
191
192         containerPoint = SwingUtilities.convertPoint(this,
193                     connecting_point, parent);
194         parent.setSecondPoint(containerPoint, this);
195
196         connected = true;
197         updateP = 2;
198       }
199
200
201     }
202     else
203     {
204       showPopupMessage("Already connected",e.getLocationOnScreen());
205     }
206     }
207     else
208     {
209       System.out.println("ERROR: Input.java: mouseClicked()");
```

```
210       }
211
212     }
213
214     public void showPopupMessage(String message, Point po)
215     {
216       DrawPanel parent = (DrawPanel) this.getParent().getParent();
217       parent.showPopupMessage(message, po);
218       tipVisable = true;
219     }
220
221     public void hidePopup()
222     {
223       DrawPanel parent = (DrawPanel) this.getParent().getParent();
224       parent.hidePopupMessage();
225       tipVisable = false;
226     }
227
228     public void mouseEntered(MouseEvent e) {
229       // TODO Auto-generated method stub
230       DrawPanel parent = (DrawPanel) this.getParent().getParent();
231       if(!connected && parent.isWireModus())
232       {
233         showPopupMessage("Connect", e.getLocationOnScreen());
234       }
235     }
236
237     public void mouseExited(MouseEvent e) {
238       // TODO Auto-generated method stub
239       if(tipVisable)
240       {
241         hidePopup();
242       }
243
244     }
245
246     public void mousePressed(MouseEvent e) {
247       // TODO Auto-generated method stub
248       // TODO Auto-generated method stub
249       if(this.getParent() instanceof Gate)
250       {
251       Gate parent = (Gate) this.getParent();
252       parent.mousePressed(e);
253       }
254
255     }
256
257     public void mouseReleased(MouseEvent e) {
258       // TODO Auto-generated method stub
259
260     }
261
262     @Override
263     public void mouseDragged(MouseEvent e) {
```

```
264       // TODO Auto-generated method stub
265       if(this.getParent() instanceof Gate)
266       {
267       Gate parent = (Gate) this.getParent();
268       parent.mouseDragged(e);
269       }
270     }
271
272     @Override
273     public void mouseMoved(MouseEvent e) {
274       // TODO Auto-generated method stub
275
276     }
277
278     /**
279      * Følgende metoder brukes ikke.
280      * Man kan unngå å ha med disse tomme metoder
281      * ved å arve klassen MouseAdapter.
282      */
283 }
```

10.14 Output.java

```
1  package gates;
2
3  import gui.DrawPanel;
4  import java.awt.Color;
5  import java.awt.Component;
6  import java.awt.Dimension;
7  import java.awt.Graphics;
8  import java.awt.Graphics2D;
9  import java.awt.Point;
10 import java.awt.event.MouseEvent;
11 import java.awt.event.MouseListener;
12 import java.awt.event.MouseMotionListener;
13 import java.awt.geom.GeneralPath;
14 import javax.swing.SwingUtilities;
15
16 /**
17  * Output.java
18  *
19  * @author Kandeeban Arumugam
20  */
21 public class Output extends WireElement implements MouseListener,
22                       MouseMotionListener {
23
24   private GeneralPath path;
25   private float x, y;
26   private int thisHeight = 6;
27   private int updateP;
28
```

```
29    private boolean connected, tipVisable;
30    private Point connecting_point, converted_point;
31    private Dimension compDim;
32    private int compSize;
33
34    /**
35     * Konstruktøren for Output
36     *
37     * @param size - Størrelsen på utgangen
38     */
39    public Output(int size) {
40      path = new GeneralPath();
41      this.compSize = size;
42      this.x = 0;
43      this.y = 3;
44      this.updateP = 0;
45      neste = null;
46
47      connected = false;
48      tipVisable = false;
49
50      connecting_point = new Point();
51      connecting_point.setLocation((x + compSize), y);
52
53      // output
54      path.moveTo(x, y);
55      path.lineTo(x + compSize, y);
56      path.closePath();
57      this.addMouseListener(this);
58      this.addMouseMotionListener(this);
59      this.compDim = new Dimension((int) path.getBounds().getWidth()
60      + 1, path.getBounds().height+ 1 + thisHeight);
61    }
62
63    public Output(){
64      path = new GeneralPath();
65      connecting_point = new Point();
66    }
67
68    public void create()
69    {
70      this.x = 0;
71      this.y = 3;
72  //    this.updateP = 0;
73  //    neste = null;
74
75  //    connected = false;
76  //    tipVisable = false;
77
78      connecting_point.setLocation((x + compSize), y);
79
80      // output
81      path.moveTo(x, y);
82      path.lineTo(x + compSize, y);
```

```
83      path.closePath();
84  //    this.addMouseListener(this);
85      this.compDim = new Dimension((int) path.getBounds().getWidth()
86      + 1, path.getBounds().height+ 1 + thisHeight);
87    }
88
89
90    public void updateNewSize()
91    {
92      path.reset();
93      connecting_point.setLocation((x + compSize), y);
94      path.moveTo(x, y);
95      path.lineTo(x + compSize, y);
96      path.closePath();
97
98      this.compDim = new Dimension((int) path.getBounds().getWidth()
99      + 1, path.getBounds().height+ 1 + thisHeight);
100
101   }
102   public void setCompDim(Dimension compDim)
103   {
104     this.compDim = compDim;
105   }
106
107   public Dimension getCompDim()
108   {
109     return this.compDim;
110   }
111
112   public void setCompSize(int compSize)
113   {
114     this.compSize = compSize;
115   }
116
117   public int getCompSize()
118   {
119     return this.compSize;
120   }
121
122   public void setConnecting_point(Point connecting_point)
123   {
124     this.connecting_point = connecting_point;
125   }
126
127   public Point getConnecting_point()
128   {
129     return this.connecting_point;
130   }
131
132   public void paintComponent(Graphics g) {
133     Graphics2D g2 = (Graphics2D) g;
134     g2.setPaint(Color.YELLOW);
135 //    g2.setPaint(Color.WHITE);
136     g2.fillRect(0, 0, getWidth(), getHeight());
```

```
137       g2.fill(path);
138       g2.setPaint(Color.BLACK);
139       g2.draw(path);
140       if(compDim != null)
141       {
142         this.setSize(compDim);
143       }
144     }
145
146     public Point getConnectionPoint(Component parent) {
147       return SwingUtilities.convertPoint(this, connecting_point, parent);
148     }
149
150     public int getLinePointY() {
151       return this.thisHeight / 2;
152     }
153
154     public Dimension getSize() {
155       return compDim;
156     }
157
158     public int getUpdatePNr()
159     {
160       return updateP;
161     }
162
163     public void setUpdatePNr(int updateP)
164     {
165       this.updateP = updateP;
166     }
167
168     public void mouseClicked(MouseEvent e) {
169       // TODO Auto-generated method stub
170       // System.out.println("Output " + e.getX() + "  " + e.getY());
171       DrawPanel parent = (DrawPanel) this.getParent().getParent();
172       if (parent.isWireModus()) {
173         if (!connected) {
174
175           if (!parent.checkConnectedFirst()) {
176             converted_point = SwingUtilities.convertPoint(this,
177                 connecting_point, parent);
178             parent.setFirstPoint(converted_point, this);
179             parent.setConnectedToOutput(true);
180             connected = true;
181             updateP = 1;
182           } else {
183
184             if (!parent.getConnectedToOutput()) {
185               parent.setConnectedToOutput(true);
186               converted_point = SwingUtilities.convertPoint(this,
187                   connecting_point, parent);
188               parent.setSecondPoint(converted_point, this);
189               System.out.println("Output 2 " + connecting_point
190                   + "   " + e.getX() + " " + e.getY());
```

```
191             connected = true;
192             updateP = 2;
193           } else {
194             showPopupMessage("Unable to connect two outputs", e
195                 .getLocationOnScreen());
196           }
197
198         }
199
200       } else {
201         showPopupMessage("Already connected", e.getLocationOnScreen());
202       }
203     } else {
204       System.out.println("ERROR:  Output.java : mouseClicked()");
205     }
206
207   }
208
209   public void showPopupMessage(String message, Point po) {
210     DrawPanel parent = (DrawPanel) this.getParent().getParent();
211     parent.showPopupMessage(message, po);
212     tipVisable = true;
213   }
214
215   public void hidePopup() {
216     DrawPanel parent = (DrawPanel) this.getParent().getParent();
217     parent.hidePopupMessage();
218     tipVisable = false;
219   }
220
221   public void mouseEntered(MouseEvent e) {
222     // TODO Auto-generated method stub
223     DrawPanel parent = (DrawPanel) this.getParent().getParent();
224     if (!connected && parent.isWireModus()) {
225       showPopupMessage("Connect", e.getLocationOnScreen());
226     }
227
228   }
229
230   public void mouseExited(MouseEvent e) {
231     // TODO Auto-generated method stub
232     if (tipVisable) {
233       hidePopup();
234     }
235   }
236
237   public void mousePressed(MouseEvent e) {
238     if(this.getParent() instanceof Gate)
239     {
240     Gate parent = (Gate) this.getParent();
241     parent.mousePressed(e);
242     }
243   }
244
```

```
245     public void mouseReleased(MouseEvent e) {
246       // TODO Auto-generated method stub
247     }
248
249     @Override
250     public void mouseDragged(MouseEvent e) {
251       // TODO Auto-generated method stub
252       if(this.getParent() instanceof Gate)
253       {
254       Gate parent = (Gate) this.getParent();
255       parent.mouseDragged(e);
256       }
257     }
258
259     /**
260      * Følgende metoder brukes ikke.
261      * Man kan unngå å ha med disse tomme metoder
262      * ved å arve klassen MouseAdapter.
263      */
264     public void mouseMoved(MouseEvent e) {
265       // TODO Auto-generated method stub
266
267     }
268 }
```

10.15 ExtendsPoint.java

```
1  package gates;
2
3  import gui.DrawPanel;
4
5  import java.awt.Color;
6  import java.awt.Graphics;
7  import java.awt.Graphics2D;
8  import java.awt.Point;
9  import java.awt.event.MouseEvent;
10 import java.awt.event.MouseListener;
11 import java.awt.event.MouseMotionListener;
12 import java.awt.geom.Ellipse2D;
13
14 import javax.swing.SwingUtilities;
15
16 /**
17  * ExtendsPoint.java - Loddings-elementet
18  *
19  * @author Kandeeban Arumugam
20  *
21  */
22 public class ExtendsPoint extends WireElement implements MouseListener,
23     MouseMotionListener {
24
```

```
25    /**
26     * Diverse variablene som brukes av denne klassen.
27     */
28    private Ellipse2D eli;
29    private WireElement venstre, hoyre;
30    private Point firstClick;
31    private ExtendsPoint selectedEx;
32    private Point connectingPoint;
33    private int vUpdateP, hUpdateP, fUpdateP;
34    static int threadNr = 1;
35    private final int threadPriority = Thread.NORM_PRIORITY;
36    public Thread t, t2;
37
38    /**
39     * Konstruktøren for ExtendsPoint
40     *
41     * @param WIREID - en unik-nøkkel.
42     */
43    public ExtendsPoint(int WIREID) {
44
45      vUpdateP = 0;
46      hUpdateP = 0;
47      fUpdateP = 0;
48
49      t = null;
50      t2 = null;
51
52      venstre = null;
53      hoyre = null;
54      selectedEx = null;
55
56      eli = new Ellipse2D.Double(0, 0, 5, 5);
57      firstClick = new Point();
58      connectingPoint = new Point();
59      this.setSize((int) eli.getWidth() + 1, (int) eli.getHeight() + 1);
60      this.addMouseListener(this);
61      this.addMouseMotionListener(this);
62    }
63
64    /**
65     * Nullstiller trådene.
66     */
67    public void reset()
68    {
69      if(t != null)
70      {
71        t.stop();
72      }
73      if(t2 != null)
74      {
75        t2.stop();
76      }
77    }
78    /******** Metodene som brukes av XMLEncoder og XMLDecoder*******/
```

```
79
80      /**
81       * En tom konstruktøren.
82       *
83       * Denne metoden gjør ingenting, derfor må set-metodene brukes,
84       * til å angi nødvendige parametere.
85       *
86       * Deretter må create()-metoden tilkalles.
87       *
88       */
89      public ExtendsPoint() {
90        t = null;
91        t2 = null;
92      }
93
94      /**
95       * Denne metoden setteropp ExtendsPoint, dersom nødvendige variablene er
96       * gitt ved å bruke set-metoder.
97       */
98      public void create() {
99        eli = new Ellipse2D.Double(0, 0, 5, 5);
100       this.setSize((int) eli.getWidth() + 1, (int) eli.getHeight() + 1);
101     }
102
103     /**
104      * Metode for å sette høye-gren.
105      *
106      * @param hoyre
107      */
108     public void setHoyre(WireElement hoyre) {
109       this.hoyre = hoyre;
110     }
111
112     /**
113      * Metoden returnerer høyre-gren.
114      * @return
115      */
116     public WireElement getHoyre() {
117       return hoyre;
118     }
119
120     /**
121      * Metoden setter venstre-gren.
122      *
123      * @param venstre
124      */
125     public void setVenstre(WireElement venstre) {
126       this.venstre = venstre;
127     }
128
129     /**
130      * Metoden returnerer venstre-gren.
131      * @return
132      */
```

```
133   public WireElement getVenstre() {
134     return venstre;
135   }
136
137   /************** END *************/
138
139   /**
140    * Metoden tegner komponenten.
141    */
142   public void paintComponent(Graphics g) {
143     Graphics2D g2 = (Graphics2D) g;
144     // g2.setPaint(Color.yellow);
145     // g2.fillRect(0, 0, 100, 100);
146     g2.fill(eli);
147     g2.setPaint(Color.BLACK);
148     g2.draw(eli);
149   }
150
151   /**
152    * Metoden for å sette igang simulasjon.
153    */
154   public synchronized void startSimulering() {
155     DrawPanel parent = (DrawPanel) this.getParent();
156
157     final int sleep = parent.getClockSpeed();
158
159     t = new Thread(new Runnable() {
160       public void run() {
161         /* VENSTRE gren i egen tråd */
162         WireElement v = venstre;
163         while (v != null) {
164           //v.registerElement(getRegisteredElement());
165           v.registerRots(rotAndForeldre, getRegisteredElement());
166           v.setSignal(signal);
167           v.setShowCurent(true);
168
169           try {
170             t.sleep(sleep);
171           } catch (InterruptedException e) {
172             // TODO Auto-generated catch block
173             e.printStackTrace();
174           }
175           v = v.neste;
176         }
177         /* END */
178
179       }
180     });
181     t.setName("ExtendsPoint " + threadNr++);
182     t.setPriority(threadPriority);
183
184     t2 = new Thread(new Runnable() {
185       public void run() {
186
```

```
187           /* Hoyre gren i egen tråd */
188           WireElement h = hoyre;
189           while (h != null) {
190             //h.registerElement(getRegisteredElement());
191             h.registerRots(rotAndForeldre, getRegisteredElement());
192             h.setSignal(signal);
193             h.setShowCurent(true);
194             try {
195               t2.sleep(sleep);
196             } catch (InterruptedException e) {
197               // TODO Auto-generated catch block
198               e.printStackTrace();
199             }
200             h = h.neste;
201           }
202           /* END */
203         }
204       });
205       t2.setName("ExtendsPoint " + threadNr++);
206       t2.setPriority(threadPriority);
207       t.start();
208       t2.start();
209
210     }
211
212     /**
213      * Metoden setter koblingspunktet for denne komponenten.
214      *
215      * @param po
216      */
217     public void setConnectingPoint(Point po) {
218       this.connectingPoint = po;
219     }
220
221     /**
222      * Metoden returnerer koblingspunktet.
223      *
224      * @return
225      */
226     public Point getConnectingPoint() {
227       return this.connectingPoint;
228     }
229
230     /**
231      * Metoden for å dra elementet.
232      */
233     public void mouseDragged(MouseEvent e) {
234       // TODO Auto-generated method stub
235       DrawPanel parent = (DrawPanel) this.getParent();
236       if (selectedEx instanceof ExtendsPoint && !parent.isWireModus()) {
237         Point mouse = e.getPoint();
238         Point new_m = SwingUtilities.convertPoint(this, mouse, parent);
239
240         int xm1 = (int) firstClick.getX();
```

```
241         int ym1 = (int) firstClick.getY();
242 
243         int xm2 = (int) new_m.getX();
244         int ym2 = (int) new_m.getY();
245 
246         int yd = (int) (ym2 - ym1);
247         int xd = (int) (xm2 - xm1);
248 
249         // Point new_connectP = new Point(selectedEx.connectingPoint.x + xd,
250         // selectedEx.connectingPoint.y + yd);
251         // if(this.connectingPoint.equals(new_connectP))
252         // {
253         // return;
254         // }
255         if (parent.isSnapToGrid()) {
256           // int gridFreq = parent.getGridFrequency();
257           // Point loc = this.getLocation();
258           // double xG = ((loc.getX() + xd) / gridFreq);
259           // double yG = ((loc.getY() + yd) / gridFreq);
260           //
261           // int xxG = (int) Math.round(xG) * gridFreq;
262           // int yyG = (int) Math.round(yG) * gridFreq;
263           // moveGateGrid(xxG, yyG);
264           // firstClick = new Point(xxG, yyG);
265         } else {
266           moveExpoPixel(xd, yd);
267           firstClick = new_m;
268         }
269 
270         parent.updateScrollBars(this);
271         parent.setSaveChanged(true);
272 
273       }
274 
275     }
276 
277     /**
278      * Metoden brukes for å flytte med angitt piksel
279      *
280      * @param xd
281      * @param yd
282      */
283     public void moveExpoPixel(int xd, int yd) {
284       Point loc = this.getLocation();
285       DrawPanel pane = (DrawPanel) this.getParent();
286 
287       Point new_connectP = new Point(selectedEx.connectingPoint.x + xd,
288           selectedEx.connectingPoint.y + yd);
289       moveWires(new_connectP);
290       selectedEx.setLocation((int) loc.getX() + xd, (int) loc.getY() + yd);
291       pane.setSaveChanged(true);
292       this.connectingPoint = new_connectP;
293     }
294 
```

```
295     /**
296      * Metoden for å flytte ledningene.
297      *
298      * @param new_connectP
299      */
300     public void moveWires(Point new_connectP) {
301       WireComponent wire1 = (WireComponent) selectedEx.getForrige();
302       WireComponent wire2 = (WireComponent) selectedEx.getHoyre();
303       WireComponent wire3 = (WireComponent) selectedEx.getVenstre();
304 
305       if (wire1 != null) {
306         if (fUpdateP == 1) {
307           Point po2 = wire1.getParentP2();
308           wire1.setWireLine(new_connectP, po2);
309         } else {
310           Point po1 = wire1.getParentP1();
311           wire1.setWireLine(po1, new_connectP);
312         }
313       }
314 
315       if (wire2 != null) {
316         if (hUpdateP == 1) {
317           Point po2 = wire2.getParentP2();
318           wire2.setWireLine(new_connectP, po2);
319         } else {
320           Point po1 = wire2.getParentP1();
321           wire2.setWireLine(po1, new_connectP);
322         }
323       }
324 
325       if (wire3 != null) {
326         if (vUpdateP == 1) {
327           Point po2 = wire3.getParentP2();
328           wire3.setWireLine(new_connectP, po2);
329         } else {
330           Point po1 = wire3.getParentP1();
331           wire3.setWireLine(po1, new_connectP);
332         }
333       }
334     }
335 
336     /**
337      * Metoden brukes for flytting
338      */
339     public void mousePressed(MouseEvent e) {
340       // TODO Auto-generated method stub
341       // System.out.println("mousePressed ");
342       DrawPanel parent = (DrawPanel) this.getParent();
343       if (!parent.isWireModus()) {
344         Point m = e.getPoint();
345         firstClick = SwingUtilities.convertPoint(this, m, parent);
346         selectedEx = this;
347         findUpdatePointers();
348       }
```

```
349     }
350
351     /**
352      * Metoden finner oppdaterings punktene til ledninger.
353      */
354     public void findUpdatePointers() {
355       WireComponent fW = (WireComponent) this.getForrige();
356       if (fW != null) {
357         if (fW.getParentP1().equals(selectedEx.getConnectingPoint())) {
358           fUpdateP = 1;
359         } else {
360           fUpdateP = 2;
361         }
362       }
363
364       WireComponent hW = (WireComponent) this.getHoyre();
365       if (hW != null) {
366         if (hW.getParentP1().equals(selectedEx.getConnectingPoint())) {
367           hUpdateP = 1;
368         } else {
369           hUpdateP = 2;
370         }
371       }
372
373       WireComponent vW = (WireComponent) this.getVenstre();
374       if (vW != null) {
375         if (vW.getParentP1().equals(selectedEx.getConnectingPoint())) {
376           vUpdateP = 1;
377         } else {
378           vUpdateP = 2;
379         }
380       }
381
382     }
383
384     /**
385      * Følgende metoder brukes ikke.
386      * Man kan unngå å ha med disse tomme metoder
387      * ved å arve klassen MouseAdapter.
388      */
389     public void mouseReleased(MouseEvent e) {
390     }
391     public void mouseExited(MouseEvent e) {
392     }
393     public void mouseClicked(MouseEvent e) {
394     }
395     public void mouseEntered(MouseEvent e) {
396     }
397     public void mouseMoved(MouseEvent e) {
398     }
399 }
```

10.16 InteractiveInput.java

```
1  package gates;
2
3  import gui.DrawPanel;
4
5  import java.awt.Color;
6  import java.awt.Graphics;
7  import java.awt.Graphics2D;
8  import java.awt.Point;
9  import java.awt.event.MouseEvent;
10 import java.awt.event.MouseListener;
11 import java.awt.event.MouseMotionListener;
12 import java.awt.geom.GeneralPath;
13
14 import javax.swing.SwingUtilities;
15
16 /**
17  * InteractiveInput.java
18  *
19  * @author Kandeeban Arumugam
20  *
21  */
22 public class InteractiveInput extends WireElement implements
23     MouseMotionListener, MouseListener, Runnable {
24
25   private GeneralPath path;
26   private GeneralPath path2;
27
28   private Output out;
29   private int outX, outY, sendSignal, updateP;
30   private Point firstClick;
31   private int elementID;
32   Thread t;
33   static int i = 1;
34   private final int threadPriority = Thread.NORM_PRIORITY;
35
36   /**
37    * Konstruktøren tar imot størrelsen og en id
38    *
39    * @param size
40    * @param elementID - en unik-nøkkel
41    */
42   public InteractiveInput(int size, int elementID) {
43     this.compSize = size;
44     this.elementID = elementID;
45
46     updateP = 0;
47     sendSignal = 0;
48
49     path = new GeneralPath();
50     path2 = new GeneralPath();
51
```

```
52      path.moveTo(0, 0);
53      path.lineTo(compSize, 0);
54      path.lineTo(compSize, compSize);
55      path.lineTo(0, compSize);
56      path.closePath();
57
58      int forhold = (compSize / 4);
59      path2.moveTo(0 + forhold, 0 + forhold);
60      path2.lineTo(compSize - forhold, 0 + forhold);
61      path2.lineTo(compSize - forhold, compSize - forhold);
62      path2.lineTo(0 + forhold, compSize - forhold);
63      path2.closePath();
64
65      out = new Output(compSize);
66      outX = compSize;
67      outY = compSize / 2;
68      out
69          .setBounds(outX, outY - 2, out.getSize().width,
70              out.getSize().height);
71
72      this.add(out);
73      this.setSize(compSize + out.getWidth(), compSize + 1);
74      this.addMouseListener(this);
75      this.addMouseMotionListener(this);
76    }
77
78    public InteractiveInput() {
79    }
80
81    public void reset() {
82      if (t != null) {
83        t.stop();
84      }
85    }
86
87    public void create() {
88      sendSignal = 0;
89      path = new GeneralPath();
90      path2 = new GeneralPath();
91
92      path.moveTo(0, 0);
93      path.lineTo(compSize, 0);
94      path.lineTo(compSize, compSize);
95      path.lineTo(0, compSize);
96      path.closePath();
97
98      int forhold = (compSize / 4);
99      path2.moveTo(0 + forhold, 0 + forhold);
100     path2.lineTo(compSize - forhold, 0 + forhold);
101     path2.lineTo(compSize - forhold, compSize - forhold);
102     path2.lineTo(0 + forhold, compSize - forhold);
103     path2.closePath();
104
105     out.create();
```

```
106       this.add(out);
107       this.setSize(compSize + out.getWidth(), compSize + 1);
108     }
109
110     public void setElementID(int elementID) {
111       this.elementID = elementID;
112     }
113
114     public int getElementID() {
115       return this.elementID;
116     }
117
118     public void setUpdateP(int updateP) {
119       this.updateP = updateP;
120     }
121
122     public int getUpdateP() {
123       return this.updateP;
124     }
125
126     public void setOut(Output out) {
127       this.out = out;
128     }
129
130     public Output getOut() {
131       return this.out;
132     }
133
134     public void setCompSize(int compSize) {
135       this.compSize = compSize;
136     }
137
138     public int getCompSize() {
139       return this.compSize;
140     }
141
142     public void paintComponent(Graphics g) {
143       Graphics2D g2 = (Graphics2D) g;
144       // g2.setPaint(Color.BLUE);
145       // g2.fillRect(0, 0, getWidth(), getHeight());
146       if (sendSignal == 0) {
147         g2.setPaint(Color.WHITE);
148         g2.fill(path2);
149       } else {
150         g2.setPaint(Color.BLACK);
151         g2.fill(path2);
152       }
153       g2.setPaint(Color.BLACK);
154       g2.draw(path);
155       g2.draw(path2);
156     }
157
158     public void mouseClicked(MouseEvent e) {
159       DrawPanel parent = (DrawPanel) this.getParent();
```

```
160       if (parent.isSimModus()) {
161         if (sendSignal == 1) {
162           sendSignal = 0;
163         } else {
164           sendSignal = 1;
165         }
166         startSimulering();
167         this.repaint();
168       }
169     }
170
171     private void startSending() {
172       DrawPanel parent = (DrawPanel) this.getParent();
173       final int sleep = parent.getClockSpeed();
174
175       WireElement wi = this.out.neste;
176       while (wi != null) {
177         wi.registerElement(getElementID());
178         wi.setSignal(sendSignal);
179         wi.setShowCurent(true);
180         Thread t = Thread.currentThread();
181         try {
182           t.sleep(sleep);
183         } catch (InterruptedException e) {
184           // TODO Auto-generated catch block
185           e.printStackTrace();
186         }
187         wi = wi.neste;
188       }
189     }
190
191     public void startSimulering() {
192       t = new Thread(this, "InteractiveInput " + i++);
193       t.setPriority(threadPriority);
194       t.start();
195     }
196
197     public void stopSimulering() {
198       if (t != null && t.isAlive()) {
199         t.stop();
200       }
201     }
202
203     public void mousePressed(MouseEvent e) {
204       // TODO Auto-generated method stub
205       Point m = e.getPoint();
206       firstClick = SwingUtilities.convertPoint(this, m, this.getParent());
207
208       WireComponent wiCom = (WireComponent) this.out.neste;
209
210       if (wiCom != null) {
211
212         if (wiCom.getParentP1().equals(
213             out.getConnectionPoint(this.getParent()))) {
```

```
214           updateP = 1;
215         } else {
216           updateP = 2;
217         }
218       }
219     }
220
221     public void run() {
222       startSending();
223     }
224
225     public void mouseDragged(MouseEvent e) {
226       DrawPanel drawP = (DrawPanel) this.getParent();
227
228       if (!drawP.isWireModus()) {
229         Point mouse = e.getPoint();
230         Point new_m = SwingUtilities.convertPoint(this, mouse, drawP);
231
232         int xm1 = (int) firstClick.getX();
233         int ym1 = (int) firstClick.getY();
234
235         int xm2 = (int) new_m.getX();
236         int ym2 = (int) new_m.getY();
237
238         int yd = (int) (ym2 - ym1);
239         int xd = (int) (xm2 - xm1);
240         Point loc = this.getLocation();
241
242         this.setLocation((int) loc.getX() + xd, (int) loc.getY() + yd);
243         firstClick = new_m;
244
245         WireComponent wiCom = (WireComponent) this.out.neste;
246         if (wiCom != null) {
247           Point p, p2;
248
249           if (updateP == 1) {
250
251             p = new Point((int) (wiCom.getParentP1().getX() + xd),
252                 (int) (wiCom.getParentP1().getY() + yd));
253             p2 = new Point((int) (wiCom.getParentP2().getX()),
254                 (int) (wiCom.getParentP2().getY()));
255             wiCom.setWireLine(p, p2);
256             drawP.setWireElement(wiCom, p);
257           } else {
258             p = new Point((int) (wiCom.getParentP1().getX()),
259                 (int) (wiCom.getParentP1().getY()));
260             p2 = new Point((int) (wiCom.getParentP2().getX() + xd),
261                 (int) (wiCom.getParentP2().getY() + yd));
262             wiCom.setWireLine(p, p2);
263             drawP.setWireElement(wiCom, p);
264           }
265         }
266
267         drawP.updateScrollBars(this);
```

```
268         drawP.setSaveChanged(true);
269       }
270     }
271
272     /**
273      * Metoden definerer formen til komponenten.
274      */
275     public boolean contains(int input_x, int input_y) {
276       int ou_x = out.getX();
277       int ou_y = out.getY();
278       int ou_w = out.getWidth();
279       int ou_h = out.getHeight();
280
281       if ((input_x <= (ou_x + ou_w) && input_x >= ou_x)
282           && (input_y <= (ou_y + ou_h) && input_y >= ou_y)) {
283         return true;
284       }
285
286       return path.contains(input_x, input_y);
287     }
288
289     /**
290      * Metoden sender videre events-ene til foreldre komponenten.
291      */
292     public void mouseMoved(MouseEvent e) {
293       DrawPanel parent = (DrawPanel) this.getParent();
294       parent.mouseMoved(e);
295     }
296
297     /**
298      * Metoden brukes ikke.
299      */
300     public void mouseReleased(MouseEvent e) {
301     }
302
303     /**
304      * Metoden brukes ikke.
305      */
306     public void mouseEntered(MouseEvent e) {
307     }
308
309     /**
310      * Metoden brukes ikke.
311      */
312     public void mouseExited(MouseEvent e) {
313     }
314     /**
315      * Følgende metoder brukes ikke.
316      * Man kan unngå å ha med disse tomme metoder
317      * ved å arve klassen MouseAdapter.
318      */
319 }
```

10.17 Led.java

```
1  package gates;
2
3  import gui.DrawPanel;
4
5  import java.awt.Color;
6  import java.awt.Graphics;
7  import java.awt.Graphics2D;
8  import java.awt.Point;
9  import java.awt.RenderingHints;
10 import java.awt.event.MouseEvent;
11 import java.awt.event.MouseListener;
12 import java.awt.event.MouseMotionListener;
13 import java.awt.geom.Ellipse2D;
14
15 import javax.swing.SwingUtilities;
16
17 /**
18  * Led.java
19  *
20  * @author Kandeeban Arumugam
21  *
22  */
23 public class Led extends WireElement implements MouseListener,
24                          MouseMotionListener{
25
26   //Diverse variablene som brukes av denne klassen
27   private Ellipse2D circle;
28   private Input inpu;
29   private int updateP;
30
31   private Point firstClick;
32   private int compSize, elementID;
33
34   /**
35    * Konstruktøren tar imot størrelsen for led-dioden
36    *
37    * @param size
38    */
39   public Led(int size)
40   {
41     this.compSize = size;
42     this.elementID = 0;
43     inpu = new Input(compSize);
44     inpu.setBounds(0, (compSize/2) - 2, inpu.getSize().width,
45                          inpu.getSize().height);
46
47     circle = new Ellipse2D.Double(compSize,0,compSize, compSize);
48
49     this.add(inpu);
50     this.setSize(compSize+inpu.getWidth(),compSize + 1);
51     this.addMouseMotionListener(this);
```

```
52      this.addMouseListener(this);
53    }
54
55    /*******Metodene som brukes av XMLEncoder og XMLDecoder********/
56    public Led()
57    {
58    }
59
60    public void create()
61    {
62      circle = new Ellipse2D.Double(compSize,0,compSize, compSize);
63      inpu.create();
64      this.add(inpu);
65      this.setSize(compSize+inpu.getWidth(),compSize + 1);
66
67    }
68
69    public void setInput(Input inpu)
70    {
71      this.inpu = inpu;
72    }
73
74    public Input getInput()
75    {
76      return this.inpu;
77    }
78
79    public void setCompSize(int compSize)
80    {
81      this.compSize = compSize;
82    }
83
84    public int getCompSize()
85    {
86      return this.compSize;
87    }
88
89    public void setElementID(int elementID)
90    {
91      this.elementID = elementID;
92    }
93
94    public int getElementID()
95    {
96      return this.elementID;
97    }
98
99    /*****************END**********************/
100
101   public void paintComponent(Graphics g) {
102     Graphics2D g2 = (Graphics2D) g;
103     g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
104                         RenderingHints.VALUE_ANTIALIAS_ON);
105     //g2.setPaint(Color.yellow);
```

```
106       //g2.fillRect(0, 0, getWidth(), getHeight());
107       //g2.fill(path);
108       if(this.signal == 1)
109       {
110         g2.setPaint(Color.RED);
111         g2.fill(circle);
112       }
113       else
114       {
115         g2.setPaint(Color.WHITE);
116         g2.fill(circle);
117       }
118       g2.setPaint(Color.BLACK);
119       g2.draw(circle);
120     }
121
122     public void mousePressed(MouseEvent e) {
123       Point m = e.getPoint();
124       firstClick = SwingUtilities.convertPoint(this, m,
125                         this.getParent());
126
127       WireComponent wiCom = (WireComponent) inpu.forrige;
128
129       if (wiCom != null) {
130
131         if (wiCom.getParentP1().equals(
132             inpu.getConnectionPoint(this.getParent()))) {
133           updateP = 1;
134         } else {
135           updateP = 2;
136         }
137       }
138     }
139
140
141     public void mouseDragged(MouseEvent e) {
142       Point mouse = e.getPoint();
143       Point new_m = SwingUtilities.convertPoint(this, mouse,
144                         this.getParent());
145
146       int xm1 = (int) firstClick.getX();
147       int ym1 = (int) firstClick.getY();
148
149       int xm2 = (int) new_m.getX();
150       int ym2 = (int) new_m.getY();
151
152       int yd = (int) (ym2 - ym1);
153       int xd = (int) (xm2 - xm1);
154       Point loc = this.getLocation();
155
156       this.setLocation((int) loc.getX() + xd, (int) loc.getY() + yd);
157       firstClick = new_m;
158
159       WireComponent wiCom = (WireComponent) inpu.forrige;
```

```
160       if (wiCom != null) {
161         DrawPanel pane = (DrawPanel) this.getParent();
162         Point p, p2;
163 
164         if (updateP == 1) {
165 
166           p = new Point((int) (wiCom.getParentP1().getX() + xd),
167               (int) (wiCom.getParentP1().getY() + yd));
168           p2 = new Point((int) (wiCom.getParentP2().getX()), (int) (wiCom
169               .getParentP2().getY()));
170           wiCom.setWireLine(p, p2);
171           pane.setWireElement(wiCom, p);
172         } else {
173           p = new Point((int) (wiCom.getParentP1().getX()), (int) (wiCom
174               .getParentP1().getY()));
175           p2 = new Point((int) (wiCom.getParentP2().getX() + xd),
176               (int) (wiCom.getParentP2().getY() + yd));
177           wiCom.setWireLine(p, p2);
178           pane.setWireElement(wiCom, p);
179         }
180       }
181 
182 
183     }
184 
185     /**
186      * Metoden sender videre events-ene til foreldre komponenten.
187      */
188     public void mouseMoved(MouseEvent e) {
189       DrawPanel parent = (DrawPanel) this.getParent();
190       parent.mouseMoved(e);
191     }
192 
193     /**
194      * Følgende metoder brukes ikke.
195      * Man kan unngå å ha med disse tomme metoder
196      * ved å arve klassen MouseAdapter.
197      */
198     public void mouseReleased(MouseEvent e) {
199     }
200     public void mouseClicked(MouseEvent e) {
201     }
202     public void mouseEntered(MouseEvent e) {
203     }
204     public void mouseExited(MouseEvent e) {
205     }
206 }
```

10.18 WireComponent.java

```
1 package gates;
```

```
2
3  import gui.DrawPanel;
4  import java.awt.BasicStroke;
5  import java.awt.Graphics;
6  import java.awt.Graphics2D;
7  import java.awt.Point;
8  import java.awt.event.MouseEvent;
9  import java.awt.event.MouseListener;
10 import java.awt.event.MouseMotionListener;
11 import java.awt.geom.GeneralPath;
12 import java.awt.geom.Line2D;
13 import javax.swing.SwingUtilities;
14
15 /**
16  * WireComponent.java
17  *
18  * @author Kandeeban Arumugam
19  */
20 public class WireComponent extends WireElement implements MouseListener,
21     MouseMotionListener {
22
23   private GeneralPath path;
24
25   private Line2D line;
26   private Point p1, p2, p1_this, p2_this, placement_P;
27   private int wireID;
28   private final int inTOinType = -100;
29   private final int outTOinType = -50;
30   private int TYPE;
31
32   private float dash1[] = { 10.0f };
33   private boolean selected, tipVisable;
34
35   /**
36    * Konstruktøren for WireComponent
37    *
38    * @param wireID - En unik id
39    */
40   public WireComponent(int wireID) {
41     path = new GeneralPath();
42     this.wireID = wireID;
43     tipVisable = false;
44
45     TYPE = 0;
46
47     selected = false;
48     line = new Line2D.Double();
49     this.addMouseListener(this);
50     this.addMouseMotionListener(this);
51   }
52
53   public WireComponent() {
54     path = new GeneralPath();
55   }
```

```
56
57    public void create() {
58      int min_x = Math.min((int) p1.getX(), (int) p2.getX());
59      int min_y = Math.min((int) p1.getY(), (int) p2.getY());
60
61      placement_P = new Point(min_x, min_y);
62
63      // differansen blir width og height
64      double x = p2.getX() - p1.getX();
65      double y = p2.getY() - p1.getY();
66
67      p1_this = new Point((int) (p1.getX() - min_x),
68          (int) (p1.getY() - min_y));
69      p2_this = new Point((int) (p2.getX() - min_x),
70          (int) (p2.getY() - min_y));
71
72      line.setLine(p1_this, p2_this);
73
74      // Dette er bare for å lage den rektangulen rundt linjen
75      if (p1_this.getX() == p2_this.getX()) {
76        path.reset();
77        path.moveTo(p1_this.getX() + 10, p1_this.getY());
78        path.lineTo(p2_this.getX() + 10, p2_this.getY());
79        path.lineTo(p2_this.getX() - 10, p2_this.getY());
80        path.lineTo(p1_this.getX() - 10, p1_this.getY());
81        path.closePath();
82      } else {
83        path.reset();
84        path.moveTo(p1_this.getX(), p1_this.getY() - 10);
85        path.lineTo(p2_this.getX(), p2_this.getY() - 10);
86        path.lineTo(p2_this.getX(), p2_this.getY() + 10);
87        path.lineTo(p1_this.getX(), p1_this.getY() + 10);
88        path.closePath();
89      }
90      int width = (int) Math.abs(x);
91      int height = (int) Math.abs(y);
92
93      this.setSize(width + 1, height + 1);
94    }
95
96    public void setWireLine(Point p1, Point p2) {
97      // parent
98      this.p1 = new Point(p1);
99      this.p2 = new Point(p2);
100
101     int min_x = Math.min((int) p1.getX(), (int) p2.getX());
102     int min_y = Math.min((int) p1.getY(), (int) p2.getY());
103
104     placement_P = new Point(min_x, min_y);
105
106     // differansen blir width og height
107     double x = p2.getX() - p1.getX();
108     double y = p2.getY() - p1.getY();
109
```

```
110       p1_this = new Point((int) (p1.getX() - min_x),
111           (int) (p1.getY() - min_y));
112       p2_this = new Point((int) (p2.getX() - min_x),
113           (int) (p2.getY() - min_y));
114
115       line.setLine(p1_this, p2_this);
116
117       // Dette er bare for å lage den rektangulen rundt linjen
118       if (p1_this.getX() == p2_this.getX()) {
119         path.reset();
120         path.moveTo(p1_this.getX() + 10, p1_this.getY());
121         path.lineTo(p2_this.getX() + 10, p2_this.getY());
122         path.lineTo(p2_this.getX() - 10, p2_this.getY());
123         path.lineTo(p1_this.getX() - 10, p1_this.getY());
124         path.closePath();
125       } else {
126         path.reset();
127         path.moveTo(p1_this.getX(), p1_this.getY() - 10);
128         path.lineTo(p2_this.getX(), p2_this.getY() - 10);
129         path.lineTo(p2_this.getX(), p2_this.getY() + 10);
130         path.lineTo(p1_this.getX(), p1_this.getY() + 10);
131         path.closePath();
132       }
133       int width = (int) Math.abs(x);
134       int height = (int) Math.abs(y);
135
136       this.setSize(width + 1, height + 1);
137       rePlacement();
138     }
139
140     public void setLine(Line2D line) {
141       this.line = line;
142     }
143
144     public Line2D getLine() {
145       return this.line;
146     }
147
148     public Point getPlacementPoint() {
149       return placement_P;
150     }
151
152     public void rePlacement() {
153       this.setLocation(placement_P);
154     }
155
156     public void showPopupMessage(String message, Point po) {
157       DrawPanel parent = (DrawPanel) this.getParent();
158       parent.showPopupMessage(message, po);
159       tipVisable = true;
160     }
161
162     public void hidePopup() {
163       DrawPanel parent = (DrawPanel) this.getParent();
```

```
164       parent.hidePopupMessage();
165       tipVisable = false;
166     }
167
168     public void setType(int type) {
169       this.TYPE = type;
170     }
171
172     public int getType() {
173       return TYPE;
174     }
175
176     public Point getParentP1() {
177       return this.p1;
178     }
179
180     public void setParentP1(Point p1) {
181       this.p1 = p1;
182     }
183
184     public Point getParentP2() {
185       return this.p2;
186     }
187
188     public void setParentP2(Point p2) {
189       this.p2 = p2;
190     }
191
192     public void paintComponent(Graphics g) {
193       Graphics2D g2 = (Graphics2D) g;
194       // g2.setPaint(Color.YELLOW);
195       // g2.fillRect(0, 0, getWidth(), getHeight());
196       g2.fill(line);
197       g2.setPaint(color);
198       // g2.draw(path);
199       if (selected) {
200
201         BasicStroke dashed = new BasicStroke(1.0f, BasicStroke.CAP_BUTT,
202             BasicStroke.JOIN_MITER, 10.0f, dash1, 0.0f);
203         g2.setStroke(dashed);
204       } else {
205         BasicStroke straight_line = new BasicStroke(1.0f);
206         g2.setStroke(straight_line);
207       }
208       g2.draw(line);
209     }
210
211     public double getStigningsTallet() {
212       return (p2_this.getY() - p1_this.getY())
213           / (p2_this.getX() - p1_this.getX());
214     }
215
216     public void setP1_this(Point p1_this) {
217       this.p1_this = p1_this;
```

```
218   }
219
220   public Point getP1_this() {
221     return this.p1_this;
222   }
223
224   public void setP2_this(Point p2_this) {
225     this.p2_this = p2_this;
226   }
227
228   public Point getP2_this() {
229     return this.p2_this;
230   }
231
232   public double getKonstantB() {
233     return p2_this.getY() - (getStigningsTallet() * p2_this.getX());
234   }
235
236   public boolean isInfinite() {
237     return Double.isInfinite(getStigningsTallet());
238   }
239
240   public int getYPoint(int x) {
241     return (int) ((getStigningsTallet() * x) + getKonstantB());
242   }
243
244   public boolean contains(int input_x, int input_y) {
245     double a = getStigningsTallet();
246     int y = getYPoint(input_x);
247
248     if ((input_x >= p1_this.getX() && input_x <= p2_this.getX())
249         && y == input_y) {
250       return true;
251     }
252
253     if (path.contains(input_x, input_y)) {
254       return true;
255     }
256
257     if (Double.isInfinite(a)) {
258       int max = 0;
259       int min = 0;
260       if (p2_this.getY() > p1_this.getY()) {
261         max = (int) p2_this.getY();
262         min = (int) p1_this.getY();
263       } else {
264         max = (int) p1_this.getY();
265         min = (int) p2_this.getY();
266       }
267
268       int p1x = (int) p1_this.getX();
269       if (input_y <= max && input_y >= min) {
270         if (p1x == input_x) {
271           return true;
```

```
272         }
273       }
274     }
275
276     return false;
277   }
278
279   public void setWireID(int ID) {
280     this.wireID = ID;
281   }
282
283   public int getWireID() {
284     return wireID;
285   }
286
287   public String toString() {
288     return "WireComponent p1 " + p1 + "\tp2 " + p2 + "\tlocation: "
289         + this.getLocation();
290   }
291
292   public Point getMovePointLocation(Point po, MovePoint mov) {
293     Point p = new Point((int) po.getX() - (mov.getWidth() / 2), (int) po
294         .getY()
295         - (mov.getHeight() / 2));
296     return p;
297   }
298
299   public void mouseClicked(MouseEvent e) {
300     DrawPanel parent = (DrawPanel) this.getParent();
301     if (parent != null && parent instanceof DrawPanel
302         && !parent.isCurrentWire(wireID)) {
303       if (parent.isWireModus()) {
304
305         if (this.getType() == outTOinType
306             && parent.getConnectedToOutput()) {
307           showPopupMessage("Unable to connect two outputs", e
308               .getLocationOnScreen());
309           return;
310         }
311
312         if (this.getType() == inTOinType
313             && parent.isConnectedToExtendsPoint()) {
314           WireElement wi = this.neste;
315           while (wi != null && !(wi instanceof ExtendsPoint)) {
316             wi = wi.neste;
317           }
318
319           if (wi instanceof ExtendsPoint) {
320             showPopupMessage("This option is disabled.", e
321                 .getLocationOnScreen());
322             return;
323           }
324
325           WireElement wi2 = this.forrige;
```

```
326             while (wi2 != null && !(wi2 instanceof ExtendsPoint)) {
327               wi2 = wi2.forrige;
328             }
329             if (wi2 instanceof ExtendsPoint) {
330               showPopupMessage("This option is disabled.", e
331                   .getLocationOnScreen());
332               return;
333             }
334           }
335
336           Point point;
337           int mouseY = 0;
338
339           if (isInfinite()) {
340             mouseY = e.getY();
341             point = new Point(0, mouseY);
342           } else {
343             mouseY = getYPoint(e.getX());
344             point = new Point(e.getX(), mouseY);
345           }
346
347           Point conv_mouse_P = SwingUtilities.convertPoint(this, point,
348               parent);
349
350           // Wire Component 1
351           WireComponent newwi = new WireComponent(parent.createWireID());
352
353           // her er det viktig å sende punkter forhold til parent
354           // siden setWireLine tar varepå parent punkter hvergang
355           newwi.setWireLine(this.p1, conv_mouse_P);
356
357           // Extention Point
358           ExtendsPoint expo = new ExtendsPoint(parent.createWireID());
359           Point par_expo_location = SwingUtilities.convertPoint(this,
360               point, parent);
361           expo.setConnectingPoint(par_expo_location);
362           expo
363               .setLocation((int) (par_expo_location.getX() - (expo
364                   .getWidth() / 2)), (int) (par_expo_location
365                   .getY() - (expo.getHeight() / 2)));
366
367           // Wire component 2
368           // her er det viktig å sende punkter forhold til parent
369           // siden setWireLine tar varepå parent punkter hvergang
370           WireComponent newwi2 = new WireComponent(parent.createWireID());
371           newwi2.setWireLine(conv_mouse_P, this.p2);
372
373           newwi.setType(this.getType());
374           newwi2.setType(this.getType());
375           if (this.getType() == outTOinType) {
376             System.out.println("fungerer");
377             parent.setConnectedToOutput(true);
378           }
379
```

```
380         newwi.setLocation(newwi.getPlacementPoint());
381         newwi2.setLocation(newwi2.getPlacementPoint());
382 
383         if (this.getType() == inTOinType) {
384           System.out.println("inTOinType");
385           updateInToInTypePointer(parent, expo, newwi, newwi2);
386         } else {
387           // Oppdaterer pekere
388           // denne metoden sjekker om hvilken linje som skal have på
389           // hvilken
390           // side av extendsPoint(venstre, høyre)
391           checkRigtLeftandUpdatePointer(parent, newwi, newwi2, expo);
392         }
393 
394         parent.add(expo);
395         parent.add(newwi);
396         parent.add(newwi2);
397 
398         // sender til parent
399         if (!parent.checkConnectedFirst()) {
400           parent.setConnectedToExtendsPoint(true);
401           parent.setFirstPoint(conv_mouse_P, expo);
402         } else {
403           parent.setSecondPoint(conv_mouse_P, expo);
404         }
405 
406         parent.remove(this);
407         parent.updateUI();
408         parent.setSaveChanged(true);
409 
410       } else {
411         if (!selected) {
412 
413           selectWires(parent);
414         } else {
415           parent.removeMoviePoints();
416         }
417 
418         this.repaint();
419 
420       }
421 
422     }
423   }
424 
425   public void selectWires(DrawPanel parent) {
426     int antSelected = 0;
427     WireElement forste_f = this.forrige;
428     WireElement denne_f = this;
429     this.setSelected(true);
430     while (!(forste_f instanceof Input || forste_f instanceof Output
431                 || forste_f instanceof ExtendsPoint)
432         && forste_f != null) {
433       if (forste_f instanceof WireComponent) {
```

```
434         WireComponent wio = (WireComponent) forste_f;
435         WireComponent de = (WireComponent) denne_f;
436 
437         wio.setSelected(true);
438 
439         if (de.getParentP1().equals(wio.getParentP1())) {
440           MovePoint mov = new MovePoint();
441           mov
442                .setLocation(getMovePointLocation(de.getParentP1(),
443                    mov));
444           mov.setWire1(de, 1);
445           mov.setWire2(wio, 1);
446           parent.add(mov, 0);
447           parent.addMoivePoint(mov);
448           parent.repaint();
449 
450         } else if (de.getParentP1().equals(wio.getParentP2())) {
451           MovePoint mov = new MovePoint();
452           mov
453                .setLocation(getMovePointLocation(de.getParentP1(),
454                    mov));
455           mov.setWire1(de, 1);
456           mov.setWire2(wio, 2);
457           parent.add(mov, 0);
458           parent.addMoivePoint(mov);
459           parent.repaint();
460         } else {
461           MovePoint mov = new MovePoint();
462           mov
463                .setLocation(getMovePointLocation(de.getParentP2(),
464                    mov));
465           mov.setWire1(de, 2);
466           mov.setWire2(wio, 1);
467           parent.add(mov, 0);
468           parent.addMoivePoint(mov);
469           parent.repaint();
470         }
471         antSelected++;
472       }
473       denne_f = forste_f;
474       forste_f = forste_f.forrige;
475     }
476 
477     WireElement forste_n = this.neste;
478     WireElement denne_n = this;
479     while (!(forste_n instanceof Input || forste_n instanceof Output
480                 || forste_n instanceof ExtendsPoint)
481         && forste_n != null) {
482       if (forste_n instanceof WireComponent) {
483         WireComponent wio = (WireComponent) forste_n;
484         WireComponent de = (WireComponent) denne_n;
485         wio.setSelected(true);
486 
487         if (de.getParentP1().equals(wio.getParentP1())) {
```

```
488             MovePoint mov = new MovePoint();
489             mov
490                 .setLocation(getMovePointLocation(de.getParentP1(),
491                     mov));
492             mov.setWire1(de, 1);
493             mov.setWire2(wio, 1);
494             parent.add(mov, 0);
495             parent.addMoivePoint(mov);
496             parent.repaint();
497           } else if (de.getParentP1().equals(wio.getParentP2())) {
498             MovePoint mov = new MovePoint();
499             mov
500                 .setLocation(getMovePointLocation(de.getParentP1(),
501                     mov));
502             mov.setWire1(de, 1);
503             mov.setWire2(wio, 2);
504             parent.add(mov, 0);
505             parent.addMoivePoint(mov);
506             parent.repaint();
507           } else {
508             MovePoint mov = new MovePoint();
509             mov
510                 .setLocation(getMovePointLocation(de.getParentP2(),
511                     mov));
512             mov.setWire1(de, 2);
513             mov.setWire2(wio, 1);
514             parent.add(mov, 0);
515             parent.addMoivePoint(mov);
516             parent.repaint();
517           }
518           antSelected++;
519         }
520         denne_n = forste_n;
521         forste_n = forste_n.neste;
522       }
523
524       if (antSelected == 0) {
525         this.setSelected(false);
526       }
527     }
528
529     public void checkRigtLeftandUpdatePointer(DrawPanel parent,
530         WireComponent newwi, WireComponent newwi2, ExtendsPoint expo) {
531       if (this.forrige instanceof WireComponent) {
532         WireComponent wire = (WireComponent) this.forrige;
533         if (wire.getParentP1().equals(newwi.getParentP1())
534             || wire.getParentP1().equals(newwi.getParentP2())
535             || wire.getParentP2().equals(newwi.getParentP1())
536             || wire.getParentP2().equals(newwi.getParentP2())) {
537
538           updatePointer(newwi, newwi2, expo);
539         } else {
540           updatePointer(newwi2, newwi, expo);
541         }
```

```
542
543     } else {
544
545       if (this.forrige instanceof Output) {
546         Output out = (Output) this.forrige;
547         if (out.getConnectionPoint(parent).equals(newwi.getParentP1())
548             || out.getConnectionPoint(parent).equals(
549                 newwi.getParentP2())) {
550           updatePointer(newwi, newwi2, expo);
551         } else {
552           updatePointer(newwi2, newwi, expo);
553         }
554
555       } else if (this.forrige instanceof Input) {
556         Input inp = (Input) this.forrige;
557         if (inp.getConnectionPoint(parent).equals(newwi.getParentP1())
558             || inp.getConnectionPoint(parent).equals(
559                 newwi.getParentP2())) {
560           updatePointer(newwi, newwi2, expo);
561         } else {
562           updatePointer(newwi2, newwi, expo);
563         }
564
565       } else if (this.forrige instanceof ExtendsPoint) {
566         ExtendsPoint exp = (ExtendsPoint) this.forrige;
567         WireComponent vensWi = (WireComponent) exp.getVenstre();
568         WireComponent hoygWi = (WireComponent) exp.getHoyre();
569
570         if (vensWi.getParentP1().equals(newwi.getParentP1())
571             || vensWi.getParentP1().equals(newwi.getParentP2())
572             || vensWi.getParentP2().equals(newwi.getParentP1())
573             || vensWi.getParentP2().equals(newwi.getParentP2())) {
574
575           updatePointer(newwi, newwi2, expo);
576         } else if (vensWi.getParentP1().equals(newwi2.getParentP1())
577             || vensWi.getParentP1().equals(newwi2.getParentP2())
578             || vensWi.getParentP2().equals(newwi2.getParentP1())
579             || vensWi.getParentP2().equals(newwi2.getParentP2())) {
580           updatePointer(newwi2, newwi, expo);
581         } else if (hoygWi.getParentP1().equals(newwi.getParentP1())
582             || hoygWi.getParentP1().equals(newwi.getParentP2())
583             || hoygWi.getParentP2().equals(newwi.getParentP1())
584             || hoygWi.getParentP2().equals(newwi.getParentP2())) {
585
586           updatePointer(newwi, newwi2, expo);
587         } else if (hoygWi.getParentP1().equals(newwi2.getParentP1())
588             || hoygWi.getParentP1().equals(newwi2.getParentP2())
589             || hoygWi.getParentP2().equals(newwi2.getParentP1())
590             || hoygWi.getParentP2().equals(newwi2.getParentP2())) {
591           updatePointer(newwi2, newwi, expo);
592         }
593       }
594
595     }
```

```
596     }
597
598     public void updatePointer(WireComponent newwi, WireComponent newwi2,
599         ExtendsPoint expo) {
600       this.forrige.neste = newwi;
601       newwi.forrige = this.forrige;
602       newwi.neste = expo;
603       expo.forrige = newwi;
604       expo.setHoyre(newwi2);
605       newwi2.forrige = expo;
606       newwi2.neste = this.neste;
607       this.neste.forrige = newwi2;
608     }
609
610     public void updateInToInTypePointer(DrawPanel parent, ExtendsPoint expo,
611         WireComponent newwi, WireComponent newwi2) {
612       parent.setConnectedTOIntoIn(true);
613
614       expo.setHoyre(newwi);
615       newwi.forrige = expo;
616       newwi.neste = this.neste;
617       this.neste.forrige = newwi;
618       expo.setVenstre(newwi2);
619       newwi2.forrige = expo;
620       newwi2.neste = this.forrige;
621
622       WireElement f = newwi2;
623       WireElement wik = this.forrige;
624       while (!(wik instanceof Input) && !(wik instanceof ExtendsPoint)) {
625         wik.neste = wik.forrige;
626         wik = wik.forrige;
627         wik.neste.forrige = f;
628         f = wik.neste;
629       }
630
631       wik.forrige = f;
632       wik.neste = null;
633
634     }
635
636     public void updatePointer2(WireComponent newwi, WireComponent newwi2,
637         ExtendsPoint expo) {
638       this.forrige.neste = newwi;
639       newwi.forrige = this.forrige;
640       newwi.neste = expo;
641       expo.forrige = newwi;
642       // expo.setHoygre(newwi2);
643       newwi2.forrige = expo;
644       newwi2.neste = this.neste;
645       this.neste.forrige = newwi2;
646
647     }
648
649     public void setSelected(boolean value) {
```

```
650       this.selected = value;
651     }
652
653     public void mouseEntered(MouseEvent e) {
654       DrawPanel parent = (DrawPanel) this.getParent();
655       if (parent.isWireModus() && !parent.isCurrentWire(wireID)) {
656         showPopupMessage("attach", e.getLocationOnScreen());
657       }
658     }
659
660     public void mouseExited(MouseEvent arg0) {
661       if (tipVisable) {
662         hidePopup();
663       }
664
665     }
666
667     public void mousePressed(MouseEvent arg0) {
668     }
669
670     public void mouseReleased(MouseEvent arg0) {
671     }
672
673     public void mouseDragged(MouseEvent arg0) {
674     }
675
676     public void mouseMoved(MouseEvent e) {
677       DrawPanel parent = (DrawPanel) this.getParent();
678       parent.mouseMoved(e);
679     }
680
681     /**
682      * Følgende metoder brukes ikke.
683      * Man kan unngå å ha med disse tomme metoder
684      * ved å arve klassen MouseAdapter.
685      */
686
687 }
```

10.19 ConstructPanel.java

```
1  package gui;
2
3
4  import java.awt.*;
5
6  import java.awt.event.MouseListener;
7  import java.io.File;
8  import javax.swing.JInternalFrame;
9  import javax.swing.JLabel;
10 import javax.swing.JOptionPane;
```

```
11 import javax.swing.JScrollPane;
12 import javax.swing.JWindow;
13 import javax.swing.UIManager;
14 import javax.swing.event.InternalFrameEvent;
15 import javax.swing.event.InternalFrameListener;
16
17
18 public class ConstructPanel extends JInternalFrame implements
19                     InternalFrameListener  {
20   static final int xOffset = 30, yOffset = 30;
21   DrawPanel tegnePanel;
22   Container content;
23   private File saveFile;
24   private Frame parent;
25   int offset = 40;
26   JWindow toolTip;
27   JLabel tipMessage;
28
29   public ConstructPanel(String name, Frame fr) {
30     super(name, true, // resizable
31         true, // closable
32         true, // maximizable
33         true);// iconifiable
34     content = this.getContentPane();
35     this.parent = fr;
36
37     toolTip = new JWindow(parent);
38     tipMessage = new JLabel();
39     tipMessage.setHorizontalAlignment(JLabel.CENTER);
40     tipMessage.setOpaque(true);
41     tipMessage.setBackground(UIManager.getColor("ToolTip.background"));
42     tipMessage.setBorder(UIManager.getBorder("ToolTip.border"));
43       toolTip.add(tipMessage);
44
45     this.addInternalFrameListener(this);
46     this.setDefaultCloseOperation(JInternalFrame.DO_NOTHING_ON_CLOSE);
47     setSize(500, 400);
48   }
49
50   public void showToolTip(String message, Point po)
51     {
52         tipMessage.setText("␣␣" + message + "␣␣");
53         toolTip.pack();
54         toolTip.setLocation(po.x, po.y + 20);
55         toolTip.setVisible(true);
56     }
57
58   public void hideToolTip()
59   {
60     toolTip.dispose();
61   }
62
63   public void createNewDrawPanel(MouseListener mouseL)
64   {
```

```
65      tegnePanel = new DrawPanel(this);
66      tegnePanel.addMouseListener(mouseL);
67      tegnePanel.setPreferredSize(new Dimension(400,300));
68      JScrollPane jScrollPane = new JScrollPane(tegnePanel,
69          JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
70          JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
71      content.add(jScrollPane);
72    }
73
74    public void setDrawPanel(DrawPanel opened, MouseListener mouseL)
75    {
76      this.tegnePanel = opened;
77      tegnePanel.addMouseListener(mouseL);
78      JScrollPane jScrollPane = new JScrollPane(tegnePanel,
79          JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
80          JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
81      content.add(jScrollPane);
82    }
83
84    public void setSaved()
85    {
86      tegnePanel.setSaveChanged(false);
87    }
88
89    public void setSaveFile(File fi)
90    {
91      this.saveFile = fi;
92    }
93
94    public File getSaveFile()
95    {
96      return saveFile;
97    }
98    public void setClockSpeed(int sp)
99    {
100     tegnePanel.setClockSpeed(sp);
101   }
102   public void setWireModus(boolean value)
103   {
104     tegnePanel.setWireModus(value);
105   }
106
107   public void setSimModus(boolean value)
108   {
109     tegnePanel.setSimModus(value);
110   }
111
112   public boolean isSimModus()
113   {
114     return tegnePanel.isSimModus();
115   }
116
117   public boolean isWireModus()
118   {
```

```
119       return tegnePanel.isWireModus();
120     }
121
122     public boolean isCursorModus()
123     {
124       return tegnePanel.isCursorModus();
125     }
126
127     public void setCursorModus(boolean value)
128     {
129       tegnePanel.setCursorModus(value);
130     }
131
132     public void setShowGrid(boolean value)
133     {
134       tegnePanel.setShowGrid(value);
135       tegnePanel.repaint();
136     }
137
138     public void setSnapToGrid(boolean value)
139     {
140       tegnePanel.setSnapToGrid(value);
141     }
142
143     public DrawPanel getTegnePanelet()
144     {
145       return tegnePanel;
146     }
147
148     public void internalFrameClosing(InternalFrameEvent e) {
149       if(tegnePanel.isSaveChanged())
150       {
151         //Custom button text
152         Object[] options = {"Yes",
153                             "No",
154                             "Cancel"};
155           int n = JOptionPane.showOptionDialog(this,
156             "Circuit has changed.\n" +
157             "Do you want to save current circuit?",
158             "Information",
159             JOptionPane.YES_NO_CANCEL_OPTION,
160             JOptionPane.QUESTION_MESSAGE,
161             null,
162             options,
163             options[2]);
164
165           if(n == 0)
166           {
167             //save and exit
168             parent.save();
169             this.dispose();
170           }
171           else if(n == 1)
172           {
```

```
173             //Exit without save
174             this.dispose();
175           }
176           else if(n == 2)
177           {
178             //return to program, donothing
179             return;
180           }
181
182
183       }
184       else
185       {
186         this.dispose();
187       }
188     }
189
190     public void internalFrameDeactivated(InternalFrameEvent arg0) {
191     }
192
193     public void internalFrameDeiconified(InternalFrameEvent arg0) {
194     }
195
196     public void internalFrameIconified(InternalFrameEvent arg0) {
197     }
198
199     public void internalFrameOpened(InternalFrameEvent arg0) {
200     }
201     public void internalFrameActivated(InternalFrameEvent arg0) {
202     }
203     public void internalFrameClosed(InternalFrameEvent arg0) {
204     }
205
206 }
```

10.20 DrawPanel.java

```
1  package gui;
2
3  import gates.AndGate;
4  import gates.Elements;
5  import gates.ExtendsPoint;
6  import gates.Gate;
7  import gates.Input;
8  import gates.InteractiveInput;
9  import gates.Led;
10 import gates.MovePoint;
11 import gates.NandGate;
12 import gates.NorGate;
13 import gates.NotGate;
14 import gates.OrGate;
```

```
15 import gates.Output;
16 import gates.WireComponent;
17 import gates.WireElement;
18 import gates.XNorGate;
19 import gates.XOrGate;
20
21 import java.awt.Color;
22 import java.awt.Component;
23 import java.awt.Dimension;
24 import java.awt.Graphics;
25 import java.awt.Graphics2D;
26 import java.awt.Point;
27 import java.awt.event.MouseEvent;
28 import java.awt.event.MouseListener;
29 import java.awt.event.MouseMotionListener;
30 import java.awt.geom.Ellipse2D;
31 import java.awt.geom.Line2D;
32 import java.util.HashMap;
33 import java.util.LinkedList;
34 import javax.swing.JPanel;
35 import javax.swing.SwingUtilities;
36
37 public class DrawPanel extends JPanel implements MouseListener,
38     MouseMotionListener {
39
40   private Point first_point, second_point;
41   private boolean connectedFirst, connectedToOutput, inTOin,
42       connectedToextendsPoint;
43   private boolean wire_modus, sim_modus, cursor_modus, showGrid;
44   private boolean saveChanged, multiSelectionMode, snapToGrid;
45   private LinkedList<InteractiveInput> inputs;
46   private Line2D tempLine;
47   private int wireID;
48   private final int inTOinType = -100;
49   private final int outTOinType = -50;
50   private int gridFrequency;
51
52   private Output out;
53   private Input in, in2;
54   private ExtendsPoint expo, expo2;
55   private LinkedList<MovePoint> move_vec;
56   private LinkedList<Gate> selectedGates;
57
58   private HashMap<Integer, WireComponent> map;
59   private int elementIDcounter;
60   ConstructPanel consP;
61   private int clockSpeed;
62
63   public DrawPanel(ConstructPanel cons) {
64     this.consP = cons;
65
66     elementIDcounter = 1;
67     wireID = 0;
68     clockSpeed = 0;
```

```
69        wire_modus = false;
70        cursor_modus = false;
71
72        showGrid = false;
73        snapToGrid = false;
74        gridFrequency = 5;
75
76        inTOin = false;
77        saveChanged = false;
78        multiSelectionMode = false;
79
80        sim_modus = false;
81        inputs = new LinkedList<InteractiveInput>();
82        move_vec = new LinkedList<MovePoint>();
83        selectedGates = new LinkedList<Gate>();
84
85        map = new HashMap<Integer, WireComponent>();
86
87        connectedToOutput = false;
88        connectedFirst = false;
89        connectedToextendsPoint = false;
90
91        tempLine = new Line2D.Double();
92
93        this.setLayout(null);
94        this.setBackground(Color.WHITE);
95        this.addMouseListener(this);
96        this.addMouseMotionListener(this);
97    }
98
99    public void paintComponent(Graphics g) {
100       super.paintComponent(g);
101       Graphics2D g2 = (Graphics2D) g;
102       // g2.setPaint(new Color(214,211,206));
103       // g2.fill(this.getBounds());
104       g2.setPaint(Color.BLACK);
105
106       if (showGrid) {
107         for (int i = 0; i < this.getWidth(); i += 20) {
108           for (int j = 0; j < this.getHeight(); j += 20)
109             g2.draw(new Ellipse2D.Double(i, j, 1, 1));
110         }
111
112       }
113       if (first_point != null) {
114         g2.draw(tempLine);
115       }
116
117   }
118
119   public void setClockSpeed(int antHz) {
120     this.clockSpeed = antHz;
121   }
122
```

```
123   public int getClockSpeed()
124   {
125     return this.clockSpeed;
126   }
127
128   public void addToList(InteractiveInput ina) {
129     inputs.add(ina);
130   }
131
132   public int getGridFrequency() {
133     return gridFrequency;
134   }
135
136   public boolean isSnapToGrid() {
137     return snapToGrid;
138   }
139
140   public void setSnapToGrid(boolean value) {
141     if (value) {
142       gridFrequency = 5;
143
144     } else {
145       gridFrequency = 1;
146     }
147     this.snapToGrid = value;
148   }
149
150   public void setCurrentSelectedGate(Gate selected) {
151     if (multiSelectionMode) {
152       selectedGates.add(selected);
153     } else {
154
155       removeSelectedGates();
156       selectedGates.add(selected);
157     }
158   }
159
160   public void removeSelectedGates() {
161     if (selectedGates.size() > 0) {
162       for (int i = 0; i < selectedGates.size(); i++) {
163         Gate selec = (Gate) selectedGates.get(i);
164         if (selec != null)
165           selec.setSelected(false);
166       }
167       selectedGates.clear();
168     }
169   }
170
171   public void addMoivePoint(MovePoint mv) {
172     move_vec.add(mv);
173   }
174
175   public void removeMoviePoints() {
176     if (move_vec.size() > 0) {
```

```
177         for (int i = 0; i < move_vec.size(); i++) {
178           MovePoint mov = (MovePoint) move_vec.get(i);
179           mov.getWire1().setSelected(false);
180           mov.getWire2().setSelected(false);
181           this.remove(mov);
182         }
183         move_vec.clear();
184         System.out.println(move_vec.size());
185         this.repaint();
186       }
187     }
188
189     public int getTotalMoivePoints() {
190       return move_vec.size();
191     }
192
193     public void setFirstPoint(Point po1, WireElement IO) {
194       first_point = po1;
195       connectedFirst = true;
196       if (IO instanceof Output)
197         this.out = (Output) IO;
198       else if (IO instanceof ExtendsPoint) {
199         if (this.expo == null)
200           this.expo = (ExtendsPoint) IO;
201         else
202           this.expo2 = (ExtendsPoint) IO;
203       } else
204         this.in = (Input) IO;
205
206     }
207
208     public void setSecondPoint(Point po2, WireElement IO) {
209       second_point = po2;
210       if (IO instanceof Output) {
211         this.out = (Output) IO;
212         // egentlig vi kan sjekke her om forrige er
213         // koblet til output
214       } else if (IO instanceof ExtendsPoint) {
215         if (this.expo == null)
216           this.expo = (ExtendsPoint) IO;
217         else
218           this.expo2 = (ExtendsPoint) IO;
219       }
220
221       else {
222         if (in == null)
223           this.in = (Input) IO;
224         else {
225           this.in2 = (Input) IO;
226         }
227       }
228       createConnection();
229     }
230
```

```
231   public void setConnectedToOutput(boolean value) {
232     connectedToOutput = value;
233   }
234
235   public void setConnectedTOIntoIn(boolean value) {
236     this.inTOin = value;
237   }
238
239   public boolean getConnectedToOutput() {
240     return connectedToOutput;
241   }
242
243   public boolean isConnectedToExtendsPoint() {
244     return connectedToextendsPoint;
245   }
246
247   public void setConnectedToExtendsPoint(boolean value) {
248     this.connectedToextendsPoint = value;
249   }
250
251   public boolean checkConnectedFirst() {
252     return connectedFirst;
253   }
254
255   public void setShowGrid(boolean value) {
256     this.showGrid = value;
257   }
258
259   public void addORGate(int x, int y, int size) {
260     OrGate or = new OrGate(size, 2);
261     or.setBounds(calculateX(x) - (or.getSize().width / 2), calculateY(y)
262         - (or.getSize().height / 2), size, size);
263     or.setElementID(createElementID());
264     this.add(or);
265     this.updateUI();
266   }
267
268   public void addNORGate(int x, int y, int size) {
269     NorGate nor = new NorGate(size, 2);
270     nor.setBounds(calculateX(x) - (nor.getSize().width / 2), calculateY(y)
271         - (nor.getSize().height / 2), size, size);
272
273     nor.setElementID(createElementID());
274     if (!isSaveChanged())
275       setSaveChanged(true);
276
277     this.add(nor);
278     this.updateUI();
279   }
280
281   public void addXORPort(int x, int y, int size) {
282     XOrGate xor = new XOrGate(size, 2);
283     xor.setBounds(calculateX(x) - (xor.getSize().width / 2), calculateY(y)
284         - (xor.getSize().height / 2), size, size);
```

```
285
286       xor.setElementID(createElementID());
287
288       if (!isSaveChanged())
289         setSaveChanged(true);
290
291       this.add(xor);
292       this.updateUI();
293
294     }
295
296     public void addXNORGate(int x, int y, int size) {
297       XNorGate xnor = new XNorGate(size, 2);
298       xnor.setBounds(calculateX(x) - (xnor.getSize().width / 2),
299           calculateY(y) - (xnor.getSize().height / 2), size, size);
300
301       xnor.setElementID(createElementID());
302
303       if (!isSaveChanged())
304         setSaveChanged(true);
305
306       this.add(xnor);
307       this.updateUI();
308     }
309
310     public int calculateX(int x) {
311       double xG = x / gridFrequency;
312       int xxG = (int) Math.round(xG) * gridFrequency;
313       return xxG;
314     }
315
316     public int calculateY(int y) {
317       double yG = y / gridFrequency;
318       int yyG = (int) Math.round(yG) * gridFrequency;
319
320       return yyG;
321     }
322
323     public void addNotPort(int x, int y, int size) {
324
325       NotGate not = new NotGate(size, 1);
326       not.setBounds(calculateX(x) - (not.getSize().width / 2), calculateY(y)
327           - (not.getSize().height / 2), size, size);
328
329       not.setElementID(createElementID());
330       if (!isSaveChanged())
331         setSaveChanged(true);
332
333       this.add(not);
334       this.updateUI();
335
336     }
337
338     public void addAndPort(int x, int y, int size) {
```

```
339       AndGate and = new AndGate(size, 2);
340       and.setBounds(calculateX(x) - (and.getSize().width / 2), calculateY(y)
341           - (and.getSize().height / 2), size, size);
342
343       and.setElementID(createElementID());
344
345       if (!isSaveChanged())
346         setSaveChanged(true);
347
348       this.add(and);
349       this.updateUI();
350
351     }
352
353     public void addNandPort(int x, int y, int size) {
354       NandGate nand = new NandGate(size, 2);
355       nand.setBounds(calculateX(x) - (nand.getSize().width / 2),
356           calculateY(y) - (nand.getSize().height / 2), size, size);
357
358       nand.setElementID(createElementID());
359
360       if (!isSaveChanged())
361         setSaveChanged(true);
362
363       this.add(nand);
364       this.updateUI();
365
366     }
367
368     public void addInterActiveInput(int x, int y, int size) {
369       InteractiveInput interA = new InteractiveInput(size,
370                           createElementID());
371       interA.setBounds(calculateX(x) - (interA.getWidth() / 2),
372       calculateY(y)- (interA.getHeight() / 2), interA.getWidth(),
373       interA.getHeight());
374
375       if (!isSaveChanged())
376         setSaveChanged(true);
377
378       inputs.add(interA);
379       this.add(interA);
380       this.updateUI();
381     }
382
383     public void addLed(int x, int y, int size) {
384       Led led = new Led(size);
385       led.setBounds(calculateX(x) - (led.getWidth() / 2), calculateY(y)
386           - (led.getHeight() / 2), led.getWidth(), led.getHeight());
387
388       led.setElementID(createElementID());
389
390       if (!isSaveChanged())
391         setSaveChanged(true);
392
```

```
393       this.add(led);
394       this.updateUI();
395     }
396
397     public boolean isWireModus() {
398       return wire_modus;
399     }
400
401     public void setWireModus(boolean value) {
402       this.wire_modus = value;
403     }
404
405     public void setCursorModus(boolean value) {
406       this.cursor_modus = value;
407     }
408
409     public boolean isCursorModus()
410     {
411       return cursor_modus;
412     }
413
414     public void setSimModus(boolean value) {
415       this.sim_modus = value;
416
417       if (sim_modus) {
418         for (int i = 0; i < inputs.size(); i++) {
419           InteractiveInput inpu = inputs.get(i);
420           inpu.startSimulering();
421         }
422       } else {
423         for (int i = 0; i < inputs.size(); i++) {
424           InteractiveInput inpu = inputs.get(i);
425           inpu.stopSimulering();
426         }
427
428         Component[] comp = this.getComponents();
429         for (int j = 0; j < comp.length; j++) {
430           if (comp[j] instanceof Gate) {
431             Gate ga = (Gate) comp[j];
432             int antall = ga.getAntallInput();
433             Input in[] = ga.getInput();
434             for (int k = 0; k < antall; k++) {
435               in[k].setSignalRegistered(false);
436             }
437
438           } else if (comp[j] instanceof Led) {
439             Led le = (Led) comp[j];
440             le.setSignal(0);
441             le.repaint();
442           }
443           else if(comp[j] instanceof ExtendsPoint)
444           {
445             ExtendsPoint ex = (ExtendsPoint) comp[j];
446             ex.reset();
```

```
447         }
448         else if(comp[j] instanceof InteractiveInput)
449         {
450           InteractiveInput inter = (InteractiveInput) comp[j];
451           inter.reset();
452         }
453         else if (comp[j] instanceof WireElement) {
454           WireElement we = (WireElement) comp[j];
455           we.reset();
456         }
457       }
458     }
459   }
460
461   public boolean isSimModus() {
462     return sim_modus;
463   }
464
465   public void setWireElement(WireComponent wiii, Point p1) {
466     wiii.setLocation(wiii.getPlacementPoint());
467   }
468
469   public void createConnection() {
470     if (wire_modus) {
471
472       if (first_point != null && second_point != null) {
473         WireComponent wiii = new WireComponent(wireID);
474         wiii.setWireLine(first_point, second_point);
475
476         // her må vi kalle plasseringsmetoden
477         setWireElement(wiii, first_point);
478         if (connectedToOutput) {
479           wiii.setType(outTOinType);
480         }
481
482         if (in != null) {
483           leggInn(in, wiii);
484         }
485
486         if (out != null) {
487           wiii.setType(outTOinType);
488           // her kan jeg kanskje oppdatere typen
489           // til inToinType wire til OutToInType?
490           leggInn(out, wiii);
491         }
492         if (in2 != null) {
493           leggInnLast(in2, wiii);
494         }
495         if (expo != null) {
496           leggInn(expo, wiii);
497         }
498         if (expo2 != null) {
499           leggInnLast(expo2, wiii);
500         }
```

```
501
502             this.add(wiii);
503             this.repaint();
504
505             first_point = second_point = null;
506             in = null;
507             out = null;
508             in2 = null;
509             expo = expo2 = null;
510             tempLine.setLine(-1, -1, -1, -1);
511             connectedFirst = false;
512             inTOin = false;
513             connectedToOutput = false;
514             connectedToextendsPoint = false;
515             map.clear();
516             wireID++;
517           }
518
519         } else {
520           System.out.println("ikke wire modus");
521         }
522       }
523
524       public void leggInn(Elements i, WireComponent wi) {
525         if (i instanceof Input) {
526           Input inp = (Input) i;
527           if (inp.getForrige() == null) {
528             inp.setForrige(wi);
529             wi.setNeste(inp);
530           }
531
532           else {
533             WireElement wik = inp.getForrige();
534             while (wik.getForrige() != null) {
535               wik = wik.getForrige();
536             }
537             wik.setForrige(wi);
538             wi.setNeste(wik);
539           }
540
541         } else if (i instanceof ExtendsPoint) {
542           ExtendsPoint exx = (ExtendsPoint) i;
543           if (!inTOin) {
544
545             if (exx.getVenstre() == null) {
546               exx.setVenstre(wi);
547               wi.setForrige(expo);
548
549             } else {
550               WireElement wik = exx.getVenstre();
551               while (wik.getNeste() != null) {
552                 wik = wik.getNeste();
553               }
554               wik.setNeste(wi);
```

```
555             wi.setForrige(wik);
556
557           }
558
559         } else {
560
561           if (exx.getForrige() == null) {
562             exx.setForrige(wi);
563             wi.setNeste(exx);
564
565           } else {
566             WireElement wik = exx.getForrige();
567             while (wik.getForrige() != null) {
568               wik = wik.getForrige();
569             }
570             wi.setNeste(wik);
571             wik.setForrige(wi);
572           }
573
574         }
575       } else {
576         Output oup = (Output) i;
577         if (oup.getNeste() == null) {
578           oup.setNeste(wi);
579           wi.setForrige(oup);
580         } else {
581           WireElement wik = oup.getNeste();
582           while (wik.getNeste() != null) {
583             wik = wik.getNeste();
584           }
585           wik.setNeste(wi);
586           wi.setForrige(wik);
587         }
588       }
589
590     }
591
592     public void leggInnLast(Elements i, WireComponent wi) {
593       if (i instanceof Input) {
594         Input inp = (Input) i;
595         if (inp.getForrige() == null) {
596           inp.setNeste(wi);
597           wi.setForrige(inp);
598
599           WireComponent wik = wi;
600           while (wik != null) {
601             wik.setType(inTOinType);
602             if (wik.getNeste() instanceof WireComponent) {
603               wik = (WireComponent) wik.getNeste();
604             } else {
605               wik = null;
606             }
607           }
608         } else {
```

```
609           System.out.println("leggInnLast: System Error!");
610         }
611       } else if (i instanceof ExtendsPoint) {
612         ExtendsPoint exx = (ExtendsPoint) i;
613         if (exx.getForrige() == null) {
614
615           exx.setNeste(wi);
616           wi.setForrige(exx);
617
618           WireComponent wik = wi;
619           while (wik != null) {
620             wik.setType(inTOinType);
621             if (wik.getNeste() instanceof WireComponent) {
622               wik = (WireComponent) wik.getNeste();
623             } else {
624               wik = null;
625             }
626           }
627
628         } else {
629           System.out.println("leggInnLast: System Error!");
630         }
631
632       }
633
634     }
635
636     public boolean isCurrentWire(int wireID) {
637       return map.containsKey(wireID);
638     }
639
640     public void mouseClicked(MouseEvent e) {
641       if (wire_modus & first_point != null) {
642         WireComponent wiii2 = new WireComponent(wireID);
643         wiii2.setWireLine(first_point, e.getPoint());
644
645         setWireElement(wiii2, first_point);
646
647         if (connectedToOutput) {
648           wiii2.setType(outTOinType);
649         }
650
651         if (in != null) {
652           // her kan vi si first point
653           wiii2.setType(outTOinType);
654           leggInn(in, wiii2);
655         } else if (out != null) {
656           // her kan vi si first point
657           leggInn(out, wiii2);
658         } else if (expo != null) {
659           leggInn(expo, wiii2);
660         }
661
662         else {
```

```
663             System.out.println("System ERROR");
664           }
665           this.add(wiii2);
666           map.put(wireID, wiii2);
667
668           first_point.setLocation(e.getX(), e.getY());
669           this.updateUI();
670           wireID++;
671
672           setSaveChanged(true);
673
674         } else {
675           removeSelectedGates();
676           removeMoviePoints();
677         }
678       }
679
680       public void mouseEntered(MouseEvent e) {
681         // TODO Auto-generated method stub
682
683       }
684
685       public void mouseExited(MouseEvent e) {
686         // TODO Auto-generated method stub
687
688       }
689
690       public void mousePressed(MouseEvent e) {
691         // TODO Auto-generated method stub
692       }
693
694       public void mouseReleased(MouseEvent e) {
695         // TODO Auto-generated method stub
696
697       }
698
699       public void mouseDragged(MouseEvent e) {
700         // TODO Auto-generated method stub
701
702       }
703
704       public void updateScrollBars(Component comp) {
705         int xO = comp.getX() + comp.getWidth();
706         int yO = comp.getY() + comp.getHeight();
707         if (xO > this.getSize().getWidth() && yO > this.getHeight()) {
708           this.setPreferredSize(new Dimension(xO, yO));
709           this.revalidate();
710         } else if (xO > this.getSize().getWidth()) {
711           this.setPreferredSize(new Dimension(xO, (int) this.getSize()
712               .getHeight()));
713           this.revalidate();
714         } else if (yO > this.getSize().getHeight()) {
715           this.setPreferredSize(new Dimension(
716               (int) this.getSize().getWidth(), yO));
```

```
717         this.revalidate();
718       }
719
720     }
721
722     public void mouseMoved(MouseEvent e) {
723       // TODO Auto-generated method stub
724       if (wire_modus && first_point != null) {
725         // Fordi, her kan komme mouseevent fra Gate klassen.
726         // Dvs. mouseEvent fra Gate blir videre sendt til denne
727         // klassen.
728         int x = e.getX();
729         int y = e.getY();
730         if (e.getSource() instanceof Elements) {
731           Elements element = (Elements) e.getSource();
732           Point new_m = SwingUtilities.convertPoint(element,
733               e.getPoint(), this);
734           x = (int) new_m.getX();
735           y = (int) new_m.getY();
736         }
737
738         tempLine.setLine(first_point, new Point(x, y));
739         this.updateUI();
740       }
741
742     }
743
744     public void setSaveChanged(boolean value) {
745       this.saveChanged = value;
746     }
747
748     public boolean isSaveChanged() {
749       return saveChanged;
750     }
751
752     public void showPopupMessage(String message, Point po) {
753       consP.showToolTip(message, po);
754     }
755
756     public void hidePopupMessage() {
757       consP.hideToolTip();
758     }
759
760     public int createWireID() {
761       return wireID++;
762     }
763
764     public int createElementID() {
765       return elementIDcounter++;
766     }
767 }
```

10.21 Frame.java

```
1  package gui;
2
3  import gates.ExtendsPoint;
4  import gates.Gate;
5  import gates.InteractiveInput;
6  import gates.Led;
7  import gates.Macro;
8  import gates.WireComponent;
9  import java.awt.*;
10 import java.awt.event.ActionEvent;
11 import java.awt.event.ActionListener;
12 import java.awt.event.KeyEvent;
13 import java.awt.event.MouseAdapter;
14 import java.awt.event.MouseEvent;
15 import java.awt.event.MouseListener;
16 import java.io.BufferedInputStream;
17 import java.io.BufferedOutputStream;
18 import java.io.File;
19 import java.io.FileInputStream;
20 import java.io.FileNotFoundException;
21 import java.io.FileOutputStream;
22 import java.io.IOException;
23 import java.io.Serializable;
24 import java.net.URL;
25 import javax.swing.ButtonGroup;
26 import javax.swing.Icon;
27 import javax.swing.ImageIcon;
28 import javax.swing.JButton;
29 import javax.swing.JCheckBoxMenuItem;
30 import javax.swing.JDesktopPane;
31 import javax.swing.JFileChooser;
32 import javax.swing.JFrame;
33 import javax.swing.JInternalFrame;
34 import javax.swing.JMenu;
35 import javax.swing.JMenuBar;
36 import javax.swing.JMenuItem;
37 import javax.swing.JOptionPane;
38 import javax.swing.JPanel;
39 import javax.swing.JToggleButton;
40 import javax.swing.JToolBar;
41 import java.beans.*;
42 import javax.swing.KeyStroke;
43 import javax.swing.UIManager;
44 import javax.swing.UnsupportedLookAndFeelException;
45
46 public class Frame extends JFrame implements ActionListener,
47             MouseListener,Serializable {
48   JDesktopPane desktop;
49
50   private JToolBar toolBar, toolBar2, toolBar3;
51   private int documentCount, antallHertz;
```

```
52
53      ButtonGroup toggleGroup;
54      JToggleButton orbutton, norbutton, xorbutton, xnorbutton, notbutton,
55          andbutton, nandbutton, bufferbutton, interactiveinputButton,
56          ledButton;
57      JToggleButton cursorB, wireB, playButton, stopButton;
58
59      boolean andport, notport, xor, xnor, or, nor, nand, buffer,
60          interactiveinput, led;
61      boolean prev_port;
62      boolean cursor, wire, play, stop, showGrid, snapToGrid;
63
64      // Pekere
65      JToggleButton prevButton;
66      JButton newDoc, save, open, macro;
67      Icon curr_press, prev_press;
68      MouseOverFunction mouseOv;
69      ToolBar3Listner tool3L;
70
71      public Frame(String tittle) { // constructor builds GUI
72        super(tittle);
73        documentCount = 0;
74        antallHertz = 0;
75        mouseOv = new MouseOverFunction();
76        tool3L = new ToolBar3Listner();
77
78        prevButton = null;
79        prev_port = false;
80        showGrid = false;
81
82        andport = false;
83        toolBar = new JToolBar();
84        toolBar.setFloatable(false);
85
86        toolBar3 = new JToolBar();
87        toolBar3.setFloatable(false);
88        toolBar2 = new JToolBar();
89        toolBar2.setFloatable(false);
90        toolBar2.addSeparator();
91        toolBar2.setOrientation(JToolBar.VERTICAL);
92        toggleGroup = new ButtonGroup();
93
94        // first button
95        orbutton = makeNavigationToggleButton("or", "addorport", "OR",
96            "Previous");
97        toggleGroup.add(orbutton);
98        toolBar.add(orbutton);
99
100       // second button
101       norbutton = makeNavigationToggleButton("nor", "UP", "NOR", "Up");
102       toggleGroup.add(norbutton);
103       toolBar.add(norbutton);
104
105       xorbutton = makeNavigationToggleButton("xor", "UP", "XOR", "Up");
```

```
106     toggleGroup.add(xorbutton);
107     toolBar.add(xorbutton);
108 
109     xnorbutton = makeNavigationToggleButton("xnor", "UP", "XNOR", "Up");
110     toggleGroup.add(xnorbutton);
111     toolBar.add(xnorbutton);
112 
113     notbutton = makeNavigationToggleButton("not", "UP", "NOT", "Up");
114     toggleGroup.add(notbutton);
115     toolBar.add(notbutton);
116 
117     andbutton = makeNavigationToggleButton("and", "UP", "AND", "Up");
118     toggleGroup.add(andbutton);
119     toolBar.add(andbutton);
120 
121     nandbutton = makeNavigationToggleButton("nand", "UP", "NAND", "Up");
122     toggleGroup.add(nandbutton);
123     toolBar.add(nandbutton);
124 
125     interactiveinputButton = makeNavigationToggleButton("input",
126         "InteractiveInput", "Input", "InteractiveInput");
127     toggleGroup.add(interactiveinputButton);
128     toolBar.add(interactiveinputButton);
129 
130     ledButton = makeNavigationToggleButton("led", "led", "Led", "led");
131     toggleGroup.add(ledButton);
132     toolBar.add(ledButton);
133 
134     cursorB = makeNavigationToggleButton("cursor", "UP",
135                         "Cursor", "cursor");
136     cursorB.setLabel("Cursor ");
137     toggleGroup.add(cursorB);
138     wireB = makeNavigationToggleButton("wire", "UP", "wire", "wire");
139     wireB.setLabel("Wire    ");
140     toggleGroup.add(wireB);
141     playButton = makeNavigationToggleButton("play", "UP", "play", "play");
142     playButton.setLabel("Play     ");
143     toggleGroup.add(playButton);
144     stopButton = makeNavigationToggleButton("stop", "UP", "stop", "stop");
145     stopButton.setLabel("Stop    ");
146     toggleGroup.add(stopButton);
147 
148     // prev_press = cursorB.getIcon();
149     prevButton = cursorB;
150     // cursorB.setIcon(cursorB.getPressedIcon());
151 
152     toolBar2.add(cursorB);
153     toolBar2.add(wireB);
154     toolBar2.add(playButton);
155     toolBar2.add(stopButton);
156     // addButtons(toolBar);
157 
158     // Toolbar3
159     newDoc = makeNavigationJButton("toolbarButtonGraphics/general/New16",
```

```
160             "newDoc", "New", "New");
161         newDoc.setLabel("New");
162         save = makeNavigationJButton("save", "save", "save", "Save");
163         save.setLabel("Save");
164         open = makeNavigationJButton("toolbarButtonGraphics/general/Open16",
165             "open", "open", "Open");
166         open.setLabel("Open");
167         macro = makeNavigationJButton("toolbarButtonGraphics/macro", "Macro",
168             "Macro", "Macro");
169
170         toolBar3.add(newDoc);
171         toolBar3.add(save);
172         toolBar3.add(open);
173         toolBar3.add(macro);
174
175         desktop = new JDesktopPane();
176         desktop.setBackground(Color.GRAY);
177
178         Container content = this.getContentPane();
179         content.add(desktop);
180         setJMenuBar(createMenuBar());
181         createInternalFrame();
182
183         JPanel toolBarPane = new JPanel(new BorderLayout());
184         toolBarPane.add(toolBar3, BorderLayout.PAGE_START);
185         toolBarPane.add(toolBar, BorderLayout.PAGE_END);
186         content.add(toolBarPane, BorderLayout.NORTH);
187
188         content.add(toolBar2, BorderLayout.EAST);
189         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
190
191         this.pack(); // does layout of components.
192         this.setVisible(true);
193         cursorB.doClick();
194
195     }// end constructor
196
197     protected JToggleButton makeNavigationToggleButton(String imageName,
198         String actionCommand, String toolTipText, String altText) {
199       // Look for the image.
200       String imgLocation = "images/" + imageName + ".gif";
201       URL imageURL = Frame.class.getResource(imgLocation);
202
203       // Create and initialize the button.
204       JToggleButton button = new JToggleButton();
205
206       button.setActionCommand(actionCommand);
207       button.setToolTipText(toolTipText);
208       button.setFocusable(false);
209       button.addActionListener(this);
210
211       if (imageURL != null) { // image found
212         button.setIcon(new ImageIcon(imageURL, altText));
213
```

```
214       } else { // no image found
215         button.setText(altText);
216         System.err.println("Resource not found: " + imgLocation);
217       }
218
219       return button;
220     }
221
222     protected JButton makeNavigationJButton(String imageName,
223         String actionCommand, String toolTipText, String altText) {
224       // Look for the image.
225       String imgLocation = "images/" + imageName + ".gif";
226       URL imageURL = Frame.class.getResource(imgLocation);
227
228       // Create and initialize the button.
229       JButton button = new JButton();
230       button.setBorderPainted(false);
231       button.setFocusable(false);
232       button.addMouseListener(mouseOv);
233       button.addActionListener(tool3L);
234
235       if (imageURL != null) { // image found
236         button.setIcon(new ImageIcon(imageURL, altText));
237       } else { // no image found
238         button.setText(altText);
239         System.err.println("Resource not found: " + imgLocation);
240       }
241
242       return button;
243     }
244
245     protected JMenuBar createMenuBar() {
246       JMenuBar menuBar = new JMenuBar();
247       MenuListner mlistner = new MenuListner();
248
249       // Set up the line menu.
250       JMenu filemenu = new JMenu("File");
251       // menu.setMnemonic(KeyEvent.VK_D);
252       menuBar.add(filemenu);
253       JMenu editmenu = new JMenu("Edit");
254       menuBar.add(editmenu);
255       JMenu circuitmenu = new JMenu("Circuit");
256       menuBar.add(circuitmenu);
257       JMenu clockSubMenu = new JMenu("Clock Speed");
258       circuitmenu.add(clockSubMenu);
259
260       JMenu viewmenu = new JMenu("View");
261       menuBar.add(viewmenu);
262       JMenu toolsmenu = new JMenu("Tools");
263       menuBar.add(toolsmenu);
264       JMenu helpmenu = new JMenu("Help");
265       menuBar.add(helpmenu);
266
267       /*************************************************/
```

```
268         // Set up the first menu item.
269         JMenuItem menuItem = new JMenuItem("New");
270         // menuItem.setMnemonic(KeyEvent.VK_N);
271         menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,
272                 ActionEvent.ALT_MASK));
273         menuItem.setActionCommand("new");
274         menuItem.addActionListener(mlistner);
275         filemenu.add(menuItem);
276 
277         // Set up the second menu item.
278         menuItem = new JMenuItem("Save");
279         // menuItem.setMnemonic(KeyEvent.VK_Q);
280         // menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q,
281         // ActionEvent.ALT_MASK));
282         menuItem.setActionCommand("save");
283         menuItem.addActionListener(mlistner);
284         filemenu.add(menuItem);
285 
286         // Set up the nr.3 menu item.
287         menuItem = new JMenuItem("Save As...");
288         // menuItem.setMnemonic(KeyEvent.VK_Q);
289         // menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q,
290         // ActionEvent.ALT_MASK));
291         menuItem.setActionCommand("saveas");
292         menuItem.addActionListener(mlistner);
293         filemenu.add(menuItem);
294 
295         // Set up the nr.4 menu item.
296         menuItem = new JMenuItem("Quit");
297         // menuItem.setMnemonic(KeyEvent.VK_Q);
298         menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Q,
299                 ActionEvent.ALT_MASK));
300         menuItem.setActionCommand("quit");
301         menuItem.addActionListener(mlistner);
302         filemenu.add(menuItem);
303 
304         /************************************************/
305         // Set up the first helpmenu item.
306         menuItem = new JMenuItem("About");
307         // menuItem.setMnemonic(KeyEvent.VK_Q);
308         menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_0,
309                 ActionEvent.ALT_MASK));
310         menuItem.setActionCommand("about");
311         menuItem.addActionListener(mlistner);
312         helpmenu.add(menuItem);
313 
314         /************************************************/
315         // Set up the first viewmenu item.
316         menuItem = new JCheckBoxMenuItem("Show Grid");
317         menuItem.setActionCommand("showgrid");
318         menuItem.addActionListener(mlistner);
319         viewmenu.add(menuItem);
320         menuItem.doClick();
321 
```

```
322         // Set up the second viewmenu item.
323         menuItem = new JCheckBoxMenuItem("Snap to Grid");
324         menuItem.setActionCommand("snaptogrid");
325         menuItem.addActionListener(mlistner);
326         viewmenu.add(menuItem);
327 
328         /************************************************/
329         ButtonGroup but = new ButtonGroup();
330 
331         // Set up the first clockSubMenu item.
332         menuItem = new JCheckBoxMenuItem("1 Hertz");
333         menuItem.setActionCommand("1hertz");
334         menuItem.addActionListener(mlistner);
335         but.add(menuItem);
336         clockSubMenu.add(menuItem);
337 
338         // Set up the second clockSubMenu item.
339         menuItem = new JCheckBoxMenuItem("2 Hertz");
340         menuItem.setActionCommand("2hertz");
341         menuItem.addActionListener(mlistner);
342         but.add(menuItem);
343         clockSubMenu.add(menuItem);
344 
345         // Set up the third clockSubMenu item.
346         menuItem = new JCheckBoxMenuItem("5 Hertz");
347         menuItem.setActionCommand("5hertz");
348         menuItem.addActionListener(mlistner);
349         but.add(menuItem);
350         clockSubMenu.add(menuItem);
351 
352         // Set up the fourth clockSubMenu item.
353         menuItem = new JCheckBoxMenuItem("10 Hertz");
354         menuItem.setActionCommand("10hertz");
355         menuItem.addActionListener(mlistner);
356         but.add(menuItem);
357         clockSubMenu.add(menuItem);
358 
359         // Set up the fifth clockSubMenu item.
360         menuItem = new JCheckBoxMenuItem("50 Hertz");
361         menuItem.setActionCommand("50hertz");
362         menuItem.addActionListener(mlistner);
363         but.add(menuItem);
364         clockSubMenu.add(menuItem);
365 
366         // Set up the sixth clockSubMenu item.
367         menuItem = new JCheckBoxMenuItem("Not Use");
368         menuItem.setActionCommand("notuse");
369         menuItem.addActionListener(mlistner);
370         but.add(menuItem);
371         clockSubMenu.add(menuItem);
372         menuItem.doClick();
373 
374         return menuBar;
375     }
```

```
376
377   // Create a new internal frame.
378   protected void createInternalFrame() {
379     documentCount++;
380     ConstructPanel consP = new ConstructPanel("Document " + documentCount,
381         this);
382     consP.createNewDrawPanel(this);
383     consP.setVisible(true); // necessary as of 1.3
384     consP.setShowGrid(showGrid);
385     consP.setSnapToGrid(snapToGrid);
386     consP.setClockSpeed(antallHertz);
387
388     desktop.add(consP);
389
390     try {
391       consP.setSelected(true);
392     } catch (java.beans.PropertyVetoException e) {
393     }
394   }
395
396   public void createMacro(File name, Component comp[]) {
397     ConstructPanel active = (ConstructPanel) desktop.getSelectedFrame();
398     if (active == null)
399       return;
400
401     DrawPanel drP = active.getTegnePanelet();
402     DrawPanel dr_new = new DrawPanel(active);
403     for (int i = 0; i < comp.length; i++) {
404
405       if (comp[i] instanceof Gate) {
406         Gate ga = (Gate) comp[i];
407         ga.create();
408         drP.createElementID();
409       } else if (comp[i] instanceof InteractiveInput) {
410         InteractiveInput ina = (InteractiveInput) comp[i];
411         ina.create();
412         // drP.addToList(ina);
413         drP.createElementID();
414       } else if (comp[i] instanceof WireComponent) {
415         WireComponent wicom = (WireComponent) comp[i];
416         wicom.create();
417         wicom.setSignalRegistered(false);
418         drP.createWireID(); // bare for at id skal være unik
419       } else if (comp[i] instanceof Led) {
420         Led l = (Led) comp[i];
421         l.create();
422         drP.createElementID();
423       } else if (comp[i] instanceof ExtendsPoint) {
424         ExtendsPoint ex = (ExtendsPoint) comp[i];
425         ex.create();
426         ex.reset();
427       } else {
428         System.out.println("hvilket? " + comp[i]);
429       }
```

```
430         dr_new.add(comp[i]);
431       }
432       Macro mac = new Macro(dr_new);
433       mac.setLocation(100, 100);
434       drP.add(mac);
435       drP.repaint();
436     }
437
438     public void createInternalFrameToExistingFile(File name,
439                           Component comp[]) {
440       // documentCount++;
441       ConstructPanel consP = new ConstructPanel(name.toString(), this);
442       DrawPanel drP = new DrawPanel(consP);
443       for (int i = 0; i < comp.length; i++) {
444
445         if (comp[i] instanceof Gate) {
446           Gate ga = (Gate) comp[i];
447           ga.create();
448           drP.createElementID();
449         } else if (comp[i] instanceof InteractiveInput) {
450           InteractiveInput ina = (InteractiveInput) comp[i];
451           ina.create();
452           drP.addToList(ina);
453           drP.createElementID();
454         } else if (comp[i] instanceof WireComponent) {
455           WireComponent wicom = (WireComponent) comp[i];
456           wicom.create();
457           wicom.setSignalRegistered(false);
458           drP.createWireID(); // bare for at id skal være unik
459         } else if (comp[i] instanceof Led) {
460           Led l = (Led) comp[i];
461           l.create();
462           drP.createElementID();
463         } else if (comp[i] instanceof ExtendsPoint) {
464           ExtendsPoint ex = (ExtendsPoint) comp[i];
465           ex.create();
466           ex.reset();
467         } else {
468           // System.out.println("hvilket? " + comp[i]);
469         }
470         drP.add(comp[i]);
471         drP.updateScrollBars(comp[i]);
472       }
473
474       consP.setDrawPanel(drP, this);
475       consP.setSaveFile(name);
476       consP.setVisible(true); // necessary as of 1.3
477       consP.setShowGrid(showGrid);
478       consP.setSnapToGrid(snapToGrid);
479       consP.setClockSpeed(antallHertz);
480       desktop.add(consP);
481
482       try {
483         consP.setSelected(true);
```

```
484       } catch (java.beans.PropertyVetoException e) {
485       }
486
487       // return consP;
488     }
489
490     /**
491      * Create the GUI and show it. For thread safety, this method should be
492      * invoked from the event-dispatching thread.
493      */
494     private static void createAndShowGUI() {
495       // Make sure we have nice window decorations.
496       try {
497         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
498       } catch (ClassNotFoundException e) {
499         // TODO Auto-generated catch block
500         e.printStackTrace();
501       } catch (InstantiationException e) {
502         // TODO Auto-generated catch block
503         e.printStackTrace();
504       } catch (IllegalAccessException e) {
505         // TODO Auto-generated catch block
506         e.printStackTrace();
507       } catch (UnsupportedLookAndFeelException e) {
508         // TODO Auto-generated catch block
509         e.printStackTrace();
510       }
511
512       // Create and set up the window.
513       Frame frame = new Frame("Digital Circuit");
514       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
515       // Display the window.
516       frame.setSize(750, 600);
517       frame.setVisible(true);
518     }
519
520     public static void main(String[] args) {
521       createAndShowGUI();
522     }
523
524     // Quit the application.
525     protected void quit() {
526       System.exit(0);
527     }
528
529     public void mouseClicked(MouseEvent e) {
530       // TODO Auto-generated method stub
531       if (e.getSource() instanceof DrawPanel) {
532         DrawPanel cons = (DrawPanel) e.getSource();
533         if (nor) {
534           cons.addNORGate(e.getX(), e.getY(), 19);
535           cursorB.doClick();
536         } else if (or) {
537           cons.addORGate(e.getX(), e.getY(), 19);
```

```
538           cursorB.doClick();
539         } else if (andport) {
540           // ConstructPanel conss = (ConstructPanel) e.getSource();
541           cons.addAndPort(e.getX(), e.getY(), 19);
542           cursorB.doClick();
543         } else if (notport) {
544           cons.addNotPort(e.getX(), e.getY(), 12);
545           cursorB.doClick();
546         } else if (xor) {
547           cons.addXORPort(e.getX(), e.getY(), 19);
548           cursorB.doClick();
549         } else if (xnor) {
550           cons.addXNORGate(e.getX(), e.getY(), 19);
551           cursorB.doClick();
552         } else if (nand) {
553           cons.addNandPort(e.getX(), e.getY(), 19);
554           cursorB.doClick();
555         } else if (interactiveinput) {
556           cons.addInterActiveInput(e.getX(), e.getY(), 12);
557           cursorB.doClick();
558         } else if (led) {
559           cons.addLed(e.getX(), e.getY(), 12);
560           cursorB.doClick();
561         }
562       }
563 
564     }
565 
566     public void mouseEntered(MouseEvent e) {
567     }
568 
569     public void mouseExited(MouseEvent e) {
570     }
571 
572     public void mousePressed(MouseEvent e) {
573     }
574 
575     public void mouseReleased(MouseEvent arg0) {
576       // TODO Auto-generated method stub
577 
578     }
579 
580     public void changeToggleButton(JToggleButton curr_button) {
581       play = stop = wire = cursor = false;
582       or = nor = xor = xnor = notport = andport =
583                     nand = led = interactiveinput = false;
584       ConstructPanel active = (ConstructPanel) desktop.getSelectedFrame();
585       if (active != null) {
586 
587         JInternalFrame[] internalFrame = desktop.getAllFrames();
588         for (int i = 0; i < internalFrame.length; i++) {
589           ConstructPanel cons = (ConstructPanel) internalFrame[i];
590           if (active.isSimModus()) {
591             cons.setSimModus(false);
```

```
592           }
593           if (active.isWireModus()) {
594             cons.setWireModus(false);
595           }
596           if (active.isCursorModus()) {
597             cons.setCursorModus(false);
598           }
599
600         }
601
602       }
603     }
604
605     public boolean isWireModus() {
606       return wire;
607     }
608
609     public void actionPerformed(ActionEvent e) {
610       // TODO Auto-generated method stub
611       if (e.getSource() == andbutton) {
612         if (!andport) {
613           changeToggleButton(andbutton);
614           andport = true;
615         }
616       } else if (e.getSource() == notbutton) {
617         if (!notport) {
618           changeToggleButton(notbutton);
619           notport = true;
620         }
621       } else if (e.getSource() == xorbutton) {
622         if (!xor) {
623           changeToggleButton(xorbutton);
624           xor = true;
625         }
626
627       } else if (e.getSource() == orbutton) {
628         if (!or) {
629           changeToggleButton(orbutton);
630           or = true;
631         }
632
633       } else if (e.getSource() == norbutton) {
634         if (!nor) {
635           changeToggleButton(norbutton);
636           nor = true;
637         }
638
639       } else if (e.getSource() == xnorbutton) {
640         if (!xnor) {
641           changeToggleButton(xnorbutton);
642           xnor = true;
643         }
644
645       } else if (e.getSource() == nandbutton) {
```

```
646         if (!nand) {
647           changeToggleButton(nandbutton);
648           nand = true;
649         }
650 
651       } else if (e.getSource() == cursorB) {
652         if (!cursor) {
653           changeToggleButton(cursorB);
654           setCursorModus(true);
655           cursor = true;
656 
657         }
658       } else if (e.getSource() == wireB) {
659         if (!wire) {
660           changeToggleButton(wireB);
661           setWireModus(true);
662           wire = true;
663         }
664       } else if (e.getSource() == interactiveinputButton) {
665         if (!interactiveinput) {
666           changeToggleButton(interactiveinputButton);
667           interactiveinput = true;
668         }
669       } else if (e.getSource() == ledButton) {
670         if (!led) {
671           changeToggleButton(ledButton);
672           led = true;
673 
674         }
675       }
676 
677       else if (e.getSource() == playButton) {
678         if (!play) {
679           changeToggleButton(playButton);
680           setSimModus(true);
681           play = true;
682 
683         }
684       } else if (e.getSource() == stopButton) {
685         changeToggleButton(cursorB);
686       }
687 
688     }
689 
690     public void about() {
691       JOptionPane.showMessageDialog(this, "Digital Circuit v.7.5");
692     }
693 
694     public void setGrid(boolean value) {
695       JInternalFrame[] internalFrame = desktop.getAllFrames();
696       for (int i = 0; i < internalFrame.length; i++) {
697         ConstructPanel cons = (ConstructPanel) internalFrame[i];
698         cons.setShowGrid(value);
699       }
```

```
700   }
701
702   public void setCursorModus(boolean value) {
703     JInternalFrame[] internalFrame = desktop.getAllFrames();
704     for (int i = 0; i < internalFrame.length; i++) {
705       ConstructPanel cons = (ConstructPanel) internalFrame[i];
706       cons.setCursorModus(value);
707     }
708   }
709
710   public void setWireModus(boolean value) {
711     JInternalFrame[] internalFrame = desktop.getAllFrames();
712     for (int i = 0; i < internalFrame.length; i++) {
713       ConstructPanel cons = (ConstructPanel) internalFrame[i];
714       cons.setWireModus(value);
715     }
716   }
717
718   public void setSimModus(boolean value) {
719     JInternalFrame[] internalFrame = desktop.getAllFrames();
720     for (int i = 0; i < internalFrame.length; i++) {
721       ConstructPanel cons = (ConstructPanel) internalFrame[i];
722       cons.setSimModus(value);
723     }
724   }
725
726   public void setClockSpeed(int value) {
727     JInternalFrame[] internalFrame = desktop.getAllFrames();
728     for (int i = 0; i < internalFrame.length; i++) {
729       ConstructPanel cons = (ConstructPanel) internalFrame[i];
730       cons.setClockSpeed(antallHertz);
731     }
732   }
733
734   public void setSnapToGrid(boolean value) {
735     JInternalFrame[] internalFrame = desktop.getAllFrames();
736     for (int i = 0; i < internalFrame.length; i++) {
737       ConstructPanel cons = (ConstructPanel) internalFrame[i];
738       cons.setSnapToGrid(value);
739     }
740   }
741
742   public File chooseAfileToOpen() {
743     JFileChooser fileChooser = new JFileChooser();
744     int returnValue = fileChooser.showOpenDialog(this);
745     if (returnValue == JFileChooser.APPROVE_OPTION) {
746       File selectedFile = fileChooser.getSelectedFile();
747       return selectedFile;
748     }
749
750     return null;
751   }
752
753   public void save() {
```

```
754       ConstructPanel active = (ConstructPanel) desktop.getSelectedFrame();
755       if (active == null) {
756         return;
757       }
758       File saveFile = active.getSaveFile();
759       if (saveFile == null) {
760         saveFile = chooseAfileToSave();
761         checkOverwrite(saveFile);
762       }
763       if (saveFile != null) {
764         saveFile(active, saveFile);
765       }
766
767     }
768
769     public void checkOverwrite(File saveFile) {
770       if (saveFile != null) {
771         if (saveFile.exists()) {
772           int response = JOptionPane.showConfirmDialog(null,
773               "Overwrite existing file?", "Confirm Overwrite",
774               JOptionPane.OK_CANCEL_OPTION,
775               JOptionPane.QUESTION_MESSAGE);
776
777           if (response == JOptionPane.CANCEL_OPTION)
778             return;
779
780         }
781       }
782     }
783
784     public void saveAs() {
785       ConstructPanel active = (ConstructPanel) desktop.getSelectedFrame();
786       if (active == null) {
787         return;
788       }
789
790       File saveFile = chooseAfileToSave();
791       checkOverwrite(saveFile);
792
793       if (saveFile != null) {
794         saveFile(active, saveFile);
795       }
796
797     }
798
799     public void saveFile(ConstructPanel active, File saveFile) {
800
801       active.setSaveFile(saveFile);
802
803       try {
804         FileOutputStream fo = new FileOutputStream(saveFile);
805         XMLEncoder encoder = new XMLEncoder(new BufferedOutputStream(fo));
806         active.setSaved();
807         active.getTegnePanelet().removeMoviePoints();
```

```
808         encoder.writeObject(active.getTegnePanelet().getComponents());
809         // encoder.writeObject(new NewGate(40));
810         active.setTitle(saveFile.toString());
811         desktop.updateUI();
812         encoder.close();
813         fo.close();
814 
815       } catch (FileNotFoundException e1) {
816         // TODO Auto-generated catch block
817         e1.printStackTrace();
818       } catch (IOException e2) {
819         // TODO Auto-generated catch block
820         e2.printStackTrace();
821       }
822 
823     }
824 
825     public File chooseAfileToSave() {
826       JFileChooser fileChooser = new JFileChooser();
827       int returnValue = fileChooser.showSaveDialog(null);
828       if (returnValue == JFileChooser.APPROVE_OPTION) {
829         String selectedFile = fileChooser.getSelectedFile().toString();
830         if (!selectedFile.endsWith(".xml")) {
831           selectedFile += ".xml";
832         }
833         return new File(selectedFile);
834       }
835 
836       return null;
837     }
838 
839     class ToolBar3Listner implements ActionListener, Serializable {
840 
841       public void actionPerformed(ActionEvent e) {
842         // TODO Auto-generated method stub
843         if (e.getSource() == newDoc) {
844           createInternalFrame();
845         } else if (e.getSource() == save) {
846           if (!cursor) {
847             cursorB.doClick();
848           }
849           save();
850         } else if (e.getSource() == open) {
851           if (!cursor) {
852             cursorB.doClick();
853           }
854           try {
855             File openfile = chooseAfileToOpen();
856             XMLDecoder d = new XMLDecoder(new BufferedInputStream(
857                 new FileInputStream(openfile)));
858             Component comp[] = (Component[]) d.readObject();
859             createInternalFrameToExistingFile(openfile, comp);
860             d.close();
861 
```

```
862         } catch (Exception ex) {
863
864         }
865
866       } else if (e.getSource() == macro) {
867         if (!cursor) {
868           cursorB.doClick();
869         }
870         try {
871           File openfile = chooseAfileToOpen();
872           XMLDecoder d = new XMLDecoder(new BufferedInputStream(
873               new FileInputStream(openfile)));
874           Component comp[] = (Component[]) d.readObject();
875           createMacro(openfile, comp);
876           d.close();
877
878         } catch (Exception ex) {
879
880         }
881       }
882     }
883
884   }
885
886   class MenuListner implements ActionListener {
887
888     public void actionPerformed(ActionEvent e) {
889       // TODO Auto-generated method stub
890       if (e.getActionCommand() == "quit") {
891         quit();
892       } else if (e.getActionCommand() == "new") {
893         createInternalFrame();
894       } else if (e.getActionCommand() == "about") {
895         about();
896       } else if (e.getActionCommand() == "showgrid") {
897         JCheckBoxMenuItem item = (JCheckBoxMenuItem) e.getSource();
898
899         if (item.isSelected()) {
900           showGrid = true;
901           setGrid(true);
902         } else {
903           showGrid = false;
904           setGrid(false);
905         }
906
907       } else if (e.getActionCommand() == "snaptogrid") {
908         JCheckBoxMenuItem item = (JCheckBoxMenuItem) e.getSource();
909
910         if (item.isSelected()) {
911           snapToGrid = true;
912           setSnapToGrid(true);
913         } else {
914           snapToGrid = false;
915           setSnapToGrid(false);
```

```
916           }
917
918         } else if (e.getActionCommand() == "1hertz") {
919           antallHertz = 1000;
920           setClockSpeed(antallHertz);
921         } else if (e.getActionCommand() == "2hertz") {
922           antallHertz = 500;
923           setClockSpeed(antallHertz);
924         } else if (e.getActionCommand() == "5hertz") {
925           antallHertz = 200;
926           setClockSpeed(antallHertz);
927         } else if (e.getActionCommand() == "10hertz") {
928           antallHertz = 100;
929           setClockSpeed(antallHertz);
930         } else if (e.getActionCommand() == "50hertz") {
931           antallHertz = 20;
932           setClockSpeed(antallHertz);
933         } else if (e.getActionCommand() == "notuse") {
934           antallHertz = 0;
935           setClockSpeed(antallHertz);
936         } else if (e.getActionCommand() == "save") {
937           if (!cursor) {
938             cursorB.doClick();
939           }
940           save();
941         } else if (e.getActionCommand() == "saveas") {
942           if (!cursor) {
943             cursorB.doClick();
944           }
945           saveAs();
946         }
947
948       }
949
950     }
951
952     class MouseOverFunction extends MouseAdapter implements Serializable {
953       public void mouseEntered(MouseEvent e) {
954         // TODO Auto-generated method stub
955         JButton but = (JButton) e.getSource();
956         but.setBorderPainted(true);
957       }
958
959       public void mouseExited(MouseEvent e) {
960         // TODO Auto-generated method stub
961         JButton but = (JButton) e.getSource();
962         but.setBorderPainted(false);
963       }
964     }
965 }
```